

U.S. DEPARTMENT OF THE INTERIOR
BUREAU OF LAND MANAGEMENT

RECEIVED
JUN 10 1941
BUREAU OF LAND MANAGEMENT

EK-HS574-TM-001

HSC50/70 Chip Descriptions Technical Manual

Prepared by Educational Services
of
Digital Equipment Corporation

First Edition, May 1986

Digital Equipment Corporation makes no representation that use of its products with those of other manufacturers will not infringe existing or future patent rights. The descriptions contained herein do not imply the granting of a license to make use or sell equipment or software as described in this manual.

Digital Equipment Corporation assumes no responsibility or liability for the proper performance of other manufacturers' products used with its products.

Digital Equipment Corporation believes that information in this publication is accurate as of its publication date. Such information is subject to change without notice. Digital Equipment Corporation is not responsible for any inadvertent errors.

Class A Computing Devices:

NOTICE: This equipment generates, uses, and may emit radio frequency energy. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC rules for operation in a commercial environment. This equipment, when operated in a residential area, may cause interference to radio/TV communications. In such event the user (owner), at his own expense, may be required to take corrective measures.

Copyright ©1986 by Digital Equipment Corporation

All Rights Reserved.

Printed in U.S.A.

The postpaid **READER'S COMMENTS** form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DECnet	UDA50
DECUS	DECsystem-10	HSC
DIGITAL	DECSYSTEM-20	MASSBUS
PDP	RSX	RA80
UNIBUS	VMS	RA81
VT	RA60	KDA50-Q
RC25	TA78-81	
VAX	TU78-81	digital
RA80	KDH50	

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by T_EX, the typesetting system developed by Donald E. Knuth at Stanford University. T_EX is a registered trademark of the American Mathematical Society

TABLE OF CONTENTS

Preface	v
CHAPTER 1 HSC50/70 SPECIAL CHIP DESCRIPTIONS	
1.1 SERDES 10 AND 16 (DC018)	1-1
1.2 Reed Solomon Generator (DC309)	1-7
1.3 REAL TIME STATE GATE ARRAY (DC703)	1-12
1.4 DUAL DESERIALIZER (DC704)	1-15
1.5 SDI/STI RECEIVER/DRIVER (DC026)	1-20
1.6 SDI/STI ENCODER/DECODER (DC027)	1-23
CHAPTER 2 HSC50/70 PAL EQUATIONS	
2.1 PAL Equations for PALS on the HSC50 Input/Output Control Processor Module	2-1
2.2 PAL Equations for PALS on the HSC70 Input/Output Control Processor Module	2-1
2.3 PAL Equations for PALS on the HSC70 Memory Module	2-1
2.4 PAL Equations for PALS on the Data Channel Module	2-1
CHAPTER 3 HSC50 BOOT FROM LISTINGS	
CHAPTER 4 HSC70 BOOT FROM LISTINGS	
FIGURES	
1-1 SERDES Pin Layout	1-2
1-2 SERDES Function Block Diagram	1-6
1-3 Reed Solomon Generator Pinout	1-7
1-4 R-S GEN Functional Block Diagram	1-9
1-5 R-S GEN Typical Write Operation	1-10
1-6 R-S GEN Typical Read Operation	1-11
1-7 Real Time State Gate Array Pinout	1-12
1-8 Dual Deserializer Pin Layout	1-15
1-9 Dual Deserializer Functional Block Diagram	1-19
1-10 SDI/STI Receiver/Driver Pin Layout	1-20
1-11 SDI/STI Receiver/Driver Functional Block Diagram	1-22
1-12 SDI/STI Encoder/Decoder Pin Layout	1-23
1-13 SDI/STI Encoder/Decoder Functional Block Diagram	1-26
2-1 23-062J5-00	2-2
2-2 23-063J5-00	2-3
2-3 23-064J5-00	2-4
2-4 23-065J5-00	2-5
2-5 23-135K5-00	2-6
2-6 23-161K3-00	2-7
2-7 23-258J5-00	2-8
2-8 23-259J5-00	2-10

2-9	23-260J5-00	2-11
2-10	23-261J5-00	2-12
2-11	23-262J5-00	2-13
2-12	23-263J5-00	2-15
2-13	23-264J5-00	2-16
2-14	23-007K7-00	2-17
2-15	23-008K7-00	2-19
2-16	23-009K7-00	2-21
2-17	23-127K5-00	2-23
2-18	23-128K5-00	2-24
2-19	23-131K5-00	2-25
2-20	23-143K5-00	2-26
2-21	23-144K5-00	2-27
2-22	23-2495-00	2-28
2-23	23-069K5-00	2-29

TABLES

1-1	SERDES Signal Definitions and Functions	1-2
1-2	Word Length Select	1-6
1-3	Read-Solomon Generator Signal Definitions And Functions	1-8
1-4	RTS Gate Array Functional Signal Description	1-13
1-5	Dual Deserializer Signal Definitions and Functions	1-16
1-6	SDI/STI Receiver/Driver Signal Definitions and Functions	1-21
1-7	SDI/STI Encoder/Decoder Signal Definitions and Functions	1-24

Preface

This manual is a support document for the *HSC50/70 Hardware Technical Manual*. It includes the following topics:

- Descriptions of special chips used in the HSC50/70
- PAL equations for the PALs in the HSC50/70
- HSC50 Boot PROM listings
- HSC70 Boot PROM listings

All of the information contained in this manual is intended for the support level field service representative.

NOTE

Because this manual covers both the HSC50 and HSC70, the acronym HSC is used to discuss functions common to both controllers.

CHAPTER 1 HSC50/70 SPECIAL CHIP DESCRIPTIONS

1.1 SERDES 10 AND 16 (DC010)

The Serializer-Deserializer (SERDES) chip was specially designed by Digital Equipment Corporation to perform data conversion between serial and parallel data formats. The data conversion may occur at a maximum frequency of approximately 25 MHz.

The SERDES contains an automatic synchronization feature that allows it to synchronize on a serial bit stream without any external control. Using an auto-correlation technique, the SERDES detects a 12-bit pattern and uses this pattern as a synchronization point. The auto-correlation feature permits up to 3 bits of the 12-bit pattern to be in error yet still achieve correct synchronization. After this point is detected, the SERDES begins the serial to parallel conversion. There are three possible parallel word lengths: 8-bit, 10-bit (SERDES 10), and 16-bit (SERDES 16).

The SERDES has one parallel output port and two serial output ports. The parallel port and one of the serial ports are tri-state. These tri-state ports have separate external enable signals. The remaining serial port is a standard TTL totem pole output.

Figure 1-1 shows the pin layout of the SERDES, and Table 1-1 provides a functional description of the signals on the output pins. Figure 1-2 is a functional block diagram of the SERDES.

Figure 1-1 SERDES Pin Layout



Table 1-1 SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input/Output	Signal Name/Function
1	TTL G	—	TTL GROUND
2	ACD L	Output	AUTO CORRELATOR—The SERDES asserts this signal when the sync byte is detected. This signal is for test purposes only and is not used.
3	SCI H	Input	SERIAL CLOCK IN—The positive edge of this signal shifts data into the shift register.

Table 1-1 (Cont.) SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input/Output	Signal Name/Function
4	O/UR H	Output	OVER/UNDER RUN—This signal is cleared by MASTER RESET. It is asserted when a parallel transfer on the parallel port is missed. The missed parallel transfer occurs in one of the following ways: 1. In parallel-out mode (POM asserted), the SERDES asserts PRDY when the parallel output data is ready. If a corresponding DR (data received) signal is not returned within the word boundary, O/UR is asserted, indicating an overrun condition. 2. In parallel-in mode (POM negated), the SERDES asserts PRDY when it is ready to be parallel loaded. If a corresponding PLS (parallel input strobe) is not received within the word boundary, O/UR is asserted, indicating an underrun condition.
5	PRDY H	Output	PARALLEL READY—NOTE: The following assumes the tri-state ports are enabled (POTSE asserted): PRDY helps initiate and complete asynchronous handshaking routines between the parallel port on the SERDES and a corresponding parallel device. In parallel-out mode (POM asserted), the SERDES asserts POM when the parallel output word is stable on the I/O pins. However, if the sync character is present on the I/O lines, PRDY is not asserted. In parallel-in mode (POM negated), the SERDES asserts PRDY when it is ready to receive a parallel word. PRDY negates in one of the following ways: 1. If POM is asserted, a negative-to-positive transition on DR (data received) negates PRDY. 2. If POM is negated, a positive-to-negative transition PLS (parallel input strobe) negates PRDY. 3. If neither 1 or 2 above occurs, the SERDES itself drops PRDY just before the parallel data becomes invalid and latches an O/UR.
6	DR H	Input	DATA RECEIVED—In parallel-out mode (POM asserted), SERDES receives this signal from the corresponding device, indicating the device has received the data. This is part of the asynchronous parallel handshake. The handshake sequence is as follows: The SERDES asserts PRDY when the parallel output data is stable. After the user has taken the data, it asserts DR H. Upon receiving DR H, the SERDES negates PRDY.
7-14 and 27-34	I/O 15:00	Input/Output	PARALLEL DATA I/O—These lines contain parallel data and their I/O function is determined by the state of POM. Their output is enabled by the POTSE (parallel output tri-state enable) signal.
15	SDI1 H	Input	SERIAL DATA IN 1—This serial input data is clocked into the shift register if SDIC is asserted or high.

Table 1-1 (Cont.) SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input/Output	Signal Name/Function
16	SDIO H	Input	SERIAL DATA IN 0—This serial input data is clocked into the shift register if SDIC is negated or low.
17	SDIC H	INPUT	SERIAL DATA INPUT CONTROL—This signal determines which serial input is shifted into the shift register. If it is low, SDIO data is moved into the shift register. If it is high, SDI1 data is moved into the shift register.
18	POTSE L	Input	PARALLEL OUT TRI-STATE ENABLE—If negated, this signal disables all the parallel outputs (I/O <15.00>).
19	CE H	Input	COUNTER ENABLE—With POM negated, CE enables the internal bit rate counter on the next positive edge of SCI. With POM asserted, CE permits the detection of the sync pattern in the SDI stream to start the bit rate counter. The sync pattern is as follows: Preamble 0s 0000000000 Sync Pattern 111101011001100 Data XXXXXX
20	GND	—	GROUND
21	WLS0	Input	WORD LENGTH SELECT 0 and 1—These two select lines determine the length of the parallel word in or out of the SERDES. Table 1-2 describes the possible parallel word lengths.
22	WLS1	Input	
23	MR L	Input	MASTER RESET—This signal resets all the internal registers and counters.
24	I/O GND	—	I/O GROUND—Isolated ground for the I/O port output buffers (I/O <15.00>).
25	POM H	Input	PARALLEL-OUT MODE—When asserted, POM causes SERDES inputs and outputs to be serial in, serial out, and parallel out. When negated, POM causes the SERDES inputs and outputs to be parallel in and serial out.
26	PLS L	Input	PARALLEL INPUT STROBE—When the SERDES is in parallel-in mode (POM negated), PLS completes the parallel handshake. For example, the SERDES asserts PRDY, indicating it is ready to be parallel loaded. After the sending device has stable data on the I/O lines, it asserts PLS. This loads the SERDES registers and causes the SERDES to negate PRDY.
35	SDISE L	Input	SERIAL OUT TRI-STATE ENABLE—If negated, this signal disables the two serial tri-state outputs (TSSIO) and TSSCO).

Table 1-1 (Cont.) SERDES Signal Definitions and Functions

Pin Number	Pin Name	Input/Output	Signal Name/Function
36	WRC H	Output	WORD RATE CLOCK—WRC is intended to be useful in implementing the parallel port. It is decoded from the Modulo-N ($N=8$, $N=10$, $N=16$) counter. It has a 50% duty cycle and a period of N bit times. It is always asserted when the counter is equal to zero and is negated after $N/2$ bit times.
37	SDO H	Output	SERIAL DATA OUT—SDO has the same delay characteristics from SDI (0 or 1) and $I/O <15:00>$ as described in TSSIX. SDO is not a tri-state output and is intended for serially cascading two SERDES running on the same Serial Clock In (SCI).
38	TSSDO H	Output	TRI-STATE SERIAL DATA OUT—TSSDO is enabled by SOTSE (Serial Out Tri-state Enable). Data transitions on this line are carefully matched in time to positive edges of TSSCO (Tri-state Serial Clock Out). If POM is asserted, the delay from SDI0 or SDI1 to TSSDO varies with the word length selected via WLS0 and WLS1. For a word length of 16 bits, the delay is 37-bit times; a word length of 10 bits, 25-bit times; a word length of 8 bits, 21-bit times. Additionally, with POM asserted, the SERDES will be in search mode until it finds the sync byte. In the 16-bit and 10-bit modes, the SERDES shifts out the complete sync byte and any data that follows it. In the 8-bit mode, however, the sync byte is truncated by one bit. This causes the serial data train coming from TSSDO and SDO to be one bit short. However, the data that follows the sync byte is not affected. If POM is negated and PLS is asserted (loading data into the internal register), then the serial output data will be available at the serial output ports exactly 4.5 bit times after PRDY becomes asserted. From then on, serial output data will always be 4.5 bit times delayed from PRDY. Additionally, with POM negated, the serial output port is not under SERDES control until data is loaded through the parallel port and CF is asserted. If the SERDES is left in idle mode while POM is negated, any data found at SDI0 or SDI1 will be shifted to SDO and TSSDO.
39	TSSCO H	Output	TRI-STATE SERIAL CLOCK OUT—This output is a delayed version of SCI. It is delayed to match the propagation delays of TSSDO and, therefore, provides a positive-going edge very precisely matched to the transition time of the TSSIX data bits. It is a tri-state signal and is controlled by SERIAL OUTPUT TRI-STATE ENABLE (SOTSE).
40	V _{CC}	—	POWER SUPPLY—This is the +5 volts supply.

1

1



1.2 Reed Solomon Generator (DC309)

The Reed Solomon Generator (R-S GEN) is a dynamic N-MOS LSI device that implements a Reed-Solomon error correction code. The chip accepts up to 10060 bits of data in 10-bit parallel (symbol) form and generates from these symbols 170 bits (17 ten-bit symbols) of error correction code (ECC) information. The chip also contains hardware to compare 170 bits of calculated ECC with 170 bits of read ECC. This compare is accomplished via an exclusive-or operation. If the data does not compare the result allows external correction of invalid data. The algorithm is capable of correcting any number of errors contained in up to eight of the 10-bit symbols in the data field.

Figure 1-3 shows the pin layout of the Reed Solomon Generator, and Table 1-3 describes the signal names.

Figure 1-3 Reed Solomon Generator Pinout



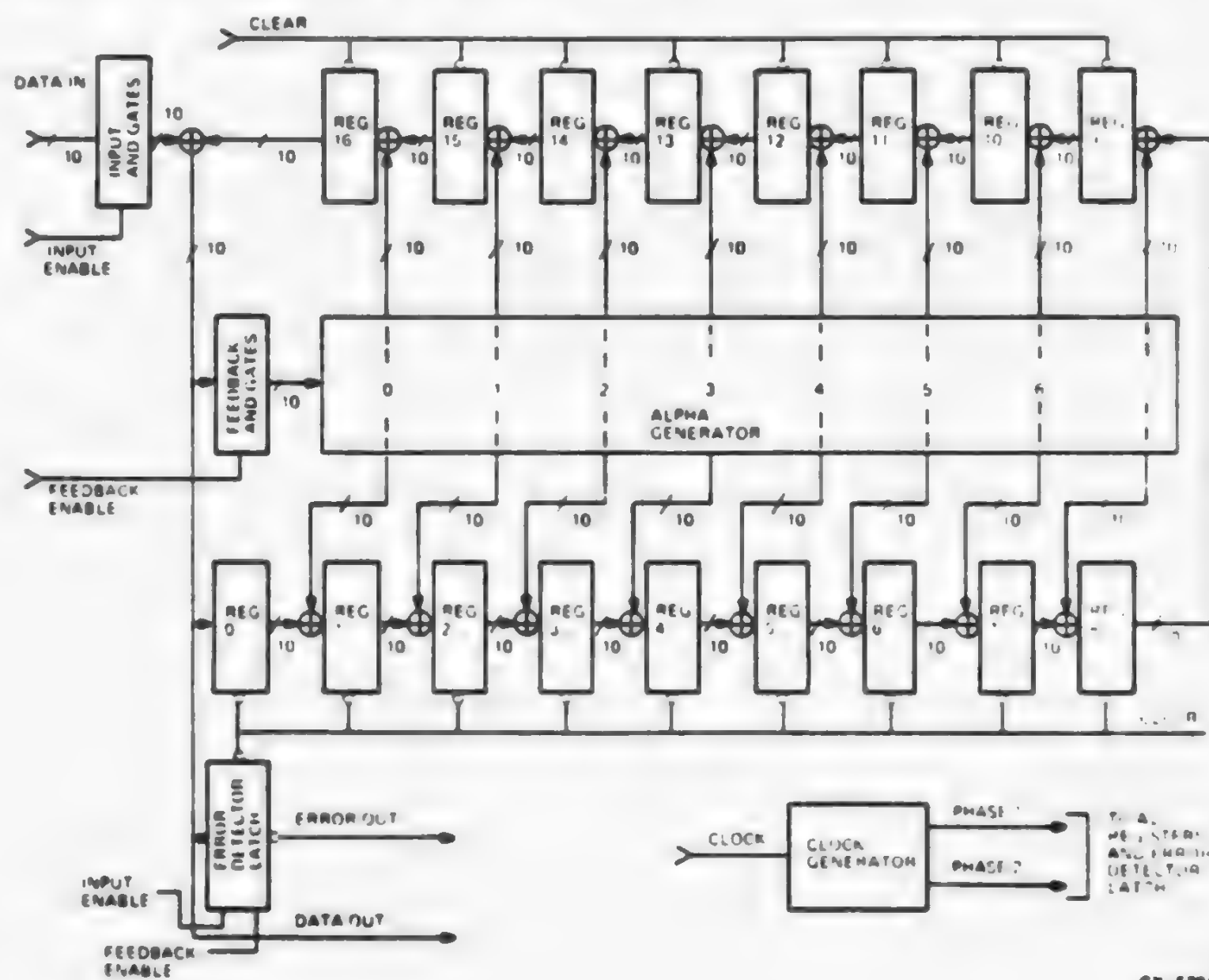
Table 1-3 Read-Solomon Generator Signal Definitions And Functions

Pin Number	Pin Name	Input/Output	Signal Name/Function
25 20 26 28 2 15 4 13 7 10	DI0 H DI1 H DI2 H DI3 H DI4 H DI5 H DI6 H DI7 H DI8 H DI9 H	Input	DATA IN <9:0>—These 10 lines are the input data lines
24 21 27 19 1 16 5 12 67 11	DO0 H DO1 H DO2 H DO3 H DO4 H DO5 H DO6 H DO7 H DO8 H DO9 H	Output	DATA OUT <9:0>—These 10 lines are the output data lines. They contain either calculated ECC or residue from an ECC comparison.
22	FBE H	Input	FEEDBACK ENABLE—FBE is asserted when ECC is calculated. It is negated when ECC is output or when ECC is compared.
23	INE H	Input	INPUT ENABLE—Input enable must be asserted when the R-S GEN is calculating ECC on data input and when the R-S GEN is calculating the compare residue on ECC input. It must be negated when the R-S GEN is outputting ECC.
17	CLR L	Input	CLEAR—This signal is synchronous with CLK H and is asserted for a full clock period. It initializes the R-S GEN.
8	CLK H	Input	CLOCK—This signal provides the timing for the R-S GEN. During normal operation, input/output signals are latched /available on the falling edge of the clock.
3	ERR L	Output	ERROR OUT—This signal is negated by CLR and asserted when the ECC residue is not equal to zero indicating an ECC error. The latch is enabled when FBE is negated and INE is asserted.

Figure 1-4 shows the R-S GEN functional block diagram. The clear signal zeros the seventeen 10-bit register stages and clears the error detector. The clock generator produces a two-phase clock for each of the registers. The following operations are performed to generate an ECC.

- The 10-bit parallel data enters through the input gate when the input enable signal is asserted
- The data passes through the feedback gate to the alpha generator when the feedback enable signal is asserted.
- The alpha generator produces eight 10-bit terms composed of three to seven input bits
- The terms are exclusive OR'd with the register outputs and the result clocked into the next register stage. This process continues until all data words are entered

Figure 1-4 R-S GEN Functional Block Diagram



At this point the ECC is stored at the outputs of the 17 register stages. To output the ECC symbols, the input and feedback enables are negated and the register data shifts out at the clock frequency

To verify input data from the disk, the R-S GEN generates an ECC based on the input data. Then, the internally generated ECC is shifted through the register stages and an exclusive-or operation is performed with the input ECC. The result goes to the output and the error generator. If the result is not zero (both ECC fields were the same), the error generator asserts the error signal.

Figure 1-5 shows the timing of a typical write operation. Figure 1-6 shows the timing of a typical read operation.

Figure 1-5 R-S GEN Typical Write Operation

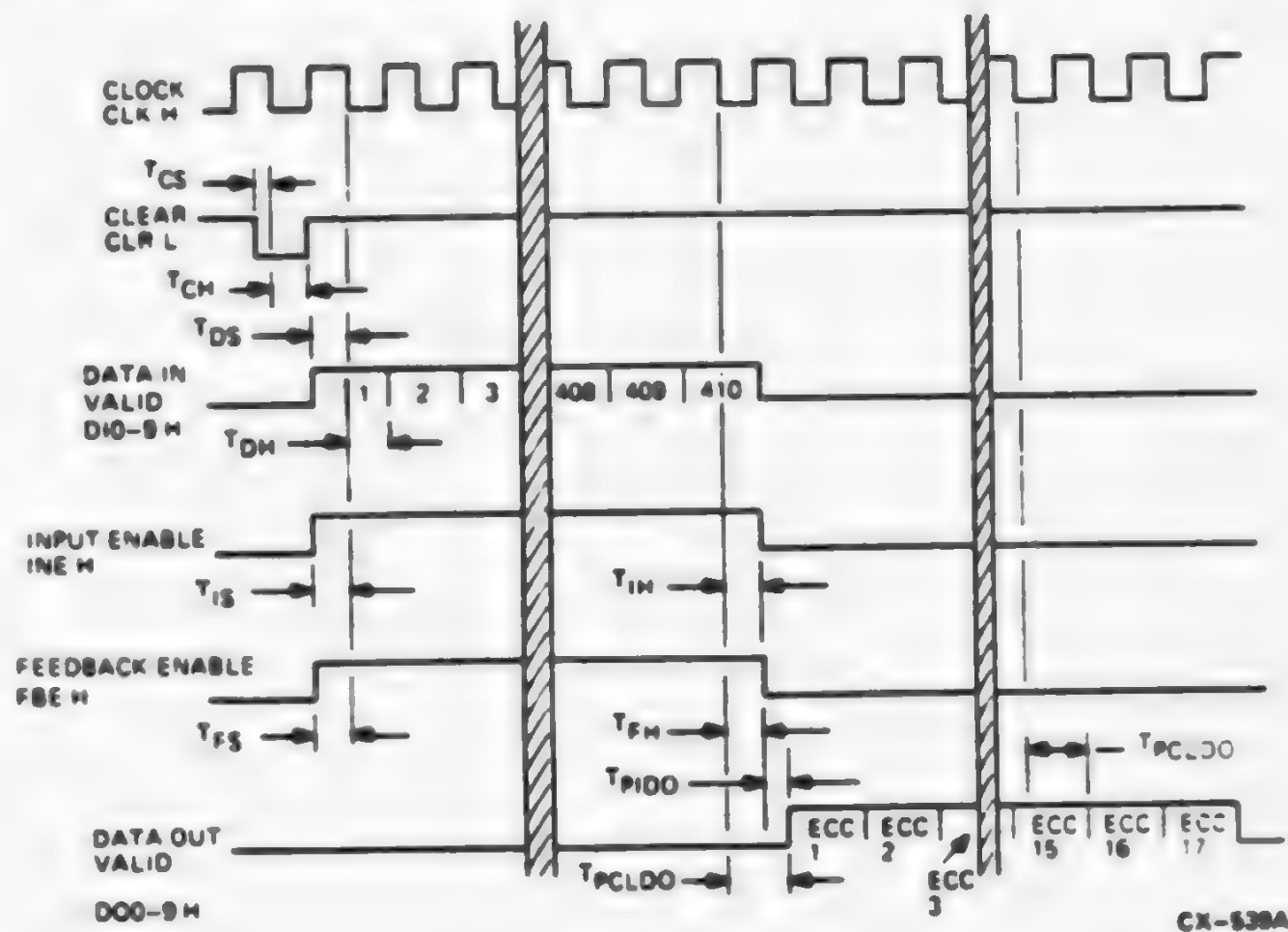
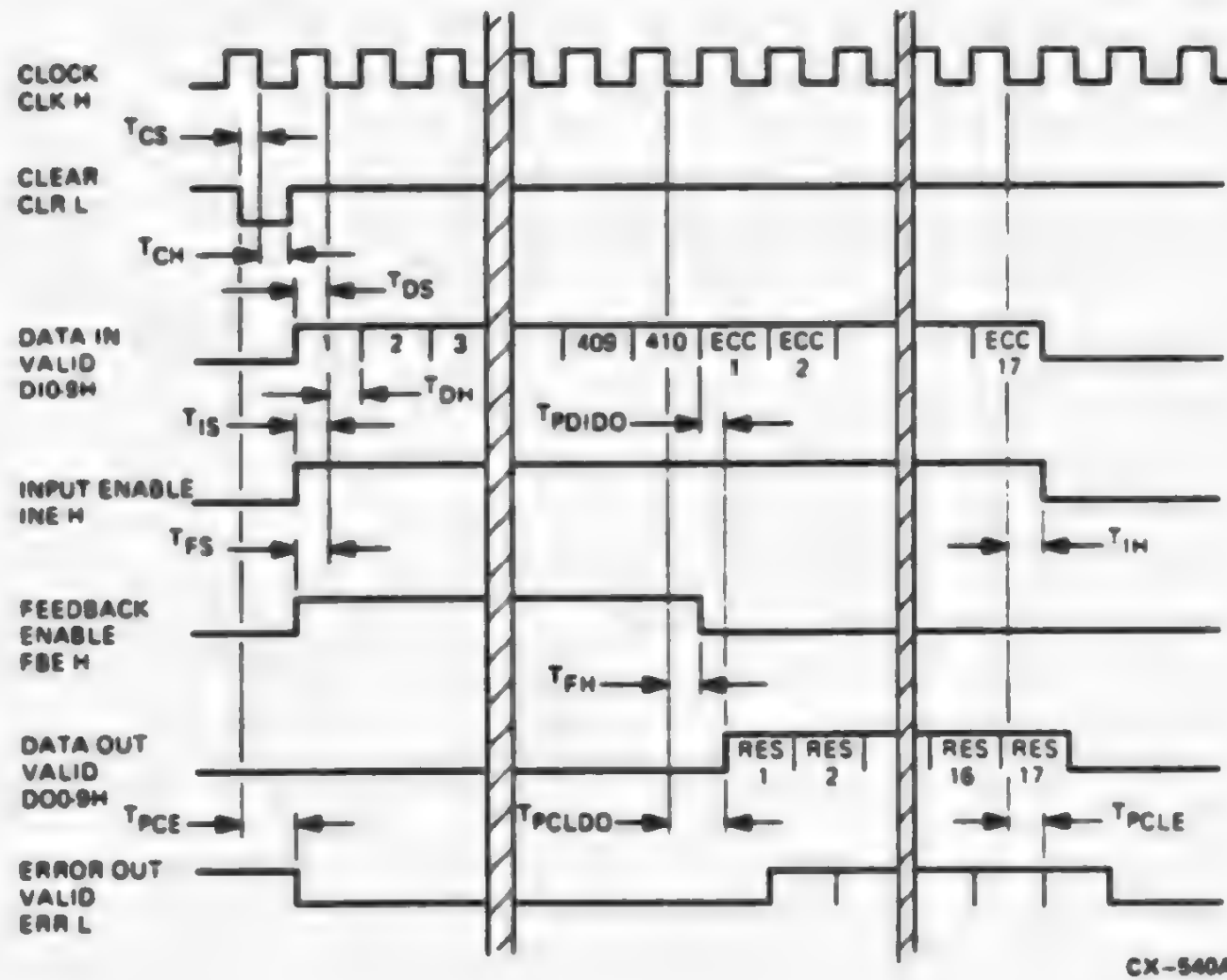


Figure 1-6 R-S GEN Typical Read Operation



1.3 REAL TIME STATE GATE ARRAY (DC703)

The Real Time State (RTS) Gate Array is a Digital Equipment Corporation-designed bipolar device for use in the Data Channel Modules. This device, in a 48-pin dual-in-line package form, provides the circuits and logic necessary to perform conversion of serial status information (and parallel information) at high data rates (minimum serial bit time = 40 nanoseconds).

The chip contains a serializer and a deserializer. The serializer converts the six asynchronous parallel status bits into a synchronous serial data stream. The data is preceded by eight zero bits (preamble) and a one (sync bit). The data bits are followed by a parity bit. The parity bit is zero when there are an odd number of ones in the six status bit, and one when there is an even number of ones in the status bits. If a parity error occurs, the parallel output buffer is not updated with the erroneous data.

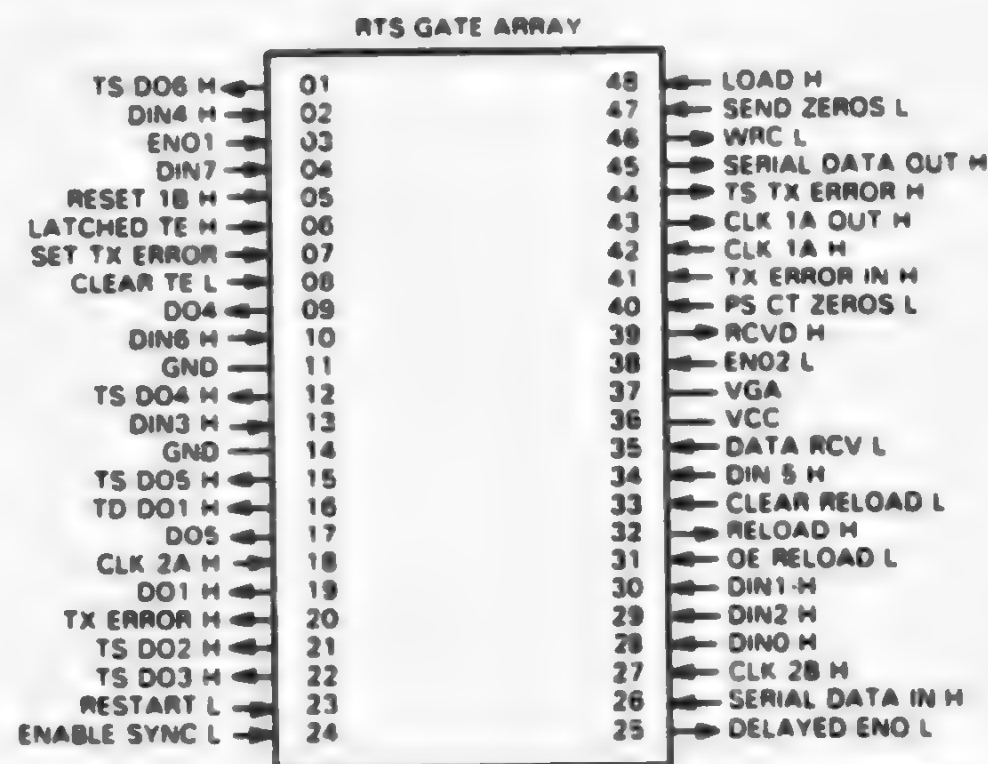
The deserializer receives a serial stream of six status bits plus a parity bit and converts them to parallel outputs. The following describes the type of parallel outputs for each bit:

- First bit—tri-state
- Second bit—tri-state and totem-pole
- Third bit—tri-state and totem-pole
- Fourth bit—tri-state
- Fifth bit—tri-state
- Sixth bit—tri-state

The deserializer looks for a sync pattern of seven or more zeros plus a one (sync bit) before it performs the serial-to-parallel conversion.

Figure 1-7 shows the real time state gate array pinout. Table 1-4 provides a functional description of the RTS gate array signals.

Figure 1-7 Real Time State Gate Array Pinout



CX-772A

Table 1-4 RTS Gate Array Functional Signal Description

Signal Name	Description
DIN <7:0> H	Parallel data inputs to the gate array. DINO H forces the gate array to send zeros on SER DATA OUT H when low, or false. When high, or true, it allows normal operation. (See also, SEND ZEROS L.) DIN7 H forces a parity error.
LOAD H	The rising edge of LOAD H provides a strobe signal that loads DIN <7:0> into the gate array input buffer.
SEND ZEROS L	Overrides DIN0 H and forces the gate array to send zeros on SER DATA OUT H.
RESET L	Causes the gate array to send zeros on SER DATA OUT H until LOAD H asserts.
OE RELOAD L	The enabling signal for RELOAD H.
SER DATA OUT H	Serial data output.
CLK 1A H	Provides a positive edge clock signal for all of the serializer synchronous logic.
CLK 1A OUT H	This is the CLK 1A H signal delayed enough to align with the SER DATA OUT H output.
CLEAR RELOAD L	Clears the RELOAD H signal.
RELOAD H	Indicates another parallel word can be loaded into the parallel data inputs of the gate array.
WRC L	WORD RATE CLOCK—When WRC is high, the gate array is shifting out the preamble zeros. When WRC is low, the gate array is shifting out SYNC 1, the RTCS data, and parity.
PS CT Zeros L	This input is only used for chip test.
CLK2A H CLK2B H	See CLK2B. Provides the positive edge triggered clock signal for all of the gate array deserializer synchronous logic. CLK2A and CLK2B are normally tied together.
RCVD H	Indicates a parallel data word is loaded in the output buffer.
ENO1 L	Output enable for the tri-state data out lines (TS DO <6:1>).
TS DO <6:1>	Parallel data outputs for the gate array. When OE1 L is asserted, these outputs are enabled.
DO1, DO4, DO5	Non tri-state outputs for data bits 1, 4, and 5.
SERIAL DIN	Serial data input.

Table 1-4 (Cont.) RTS Gate Array Functional Signal Description

Signal Name	Description
RESTART L	Asynchronously initializes the gate array.
LATCHED TE H	Latches a transmission or a parity error until the CLEAR TE L line is asserted.
TS TX Error H	This tri-state signal asserts if a parity error or a transmission error occurs.
TX Error H	Non-tri-state version of TS TX ERROR H.
CLEAR TE L	Clears TX ERROR H.
ENO2 L	Enables TS TX ERROR H and RCVD H.
DATA RCV L	Clears RCVD H.
SET TX ERROR L	Sets TX ERROR H, or asserted.
DELAYED ENO L	This is the output buffer load delayed by two clock times.
ENABLE SYNC L	When asserted, allows the output buffer to be updated

1.4 DUAL DESERIALIZER (DC704)

The Dual Deserializer Chip is a Digital Equipment Corporation- designed bipolar chip for use in the Disk Data Channel Module. This chip provides the circuits and logic necessary to perform conversion of serial status information at high data rates (minimum serial bit time is 40 nanoseconds).

This chip contains two deserializers. The deserializers receive a serial stream of six status bits plus a parity bit and converts these to parallel outputs. All six outputs are tri-state with bit four and five being TTL totem pole outputs as well. The number of ones in the six status bits plus the parity bit must always be odd. If a parity error occurs, the parallel output buffer will not be updated with erroneous data.

The deserializer looks for a sync pattern of eight or more zeros plus a one sync bit before it performs the serial-to-parallel conversion.

Figure 1-8 shows the pin layout of the dual deserializer and Table 1-5 provides a functional description of the signals on the output pins. Figure 1-9 is a functional block diagram of the dual deserializer

Figure 1-8 Dual Deserializer Pin Layout



CX-1878A

Table 1-5 Dual Deserializer Signal Definitions and Functions

Pin Number	Function	Description
1	NTS DO5A H	The non-tri-state (TTL) output bit five.
2	DATA OUT 5A H	Parallel output data for the D5A data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted.
3	DATA OUT 4 H	Parallel output data for the D4 data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted.
4	NTS DO4 H	The non-tri-state (TTL) output bit four.
5	DATA RCV L	This input clears the RCVD H output latch
6	NTS DO4A H	The non tri-state (TTL) output bit four
7	RCVD H	This TTL output indicates that a frame has just been loaded to the output buffer.
8	DATA OUT 4AS H	Parallel output data for the D4A data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted.
9	OUTPUT 2 ENABLE L	The input which enables tri-state outputs—RCVD H and TX ERROR H.
10	DATA OUT 3 H	Parallel output data for the D3 data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted.
11	GND	Ground.
12	SET TX ERROR L	This input sets the TX ERROR flip-flop to a high.
13	RCVD A H	This TTL output indicates that a frame has just been loaded to the output buffer.
14	GND	Ground.
15	DATA RCV A L	This input clears the RCVD A H output latch.
16	OUTPUT ENABLE L	The input which enables the tri-state outputs (DATA OUT 1 H, DATA OUT 2 H, DATA OUT 3 H, DATA OUT 4 H, DATA OUT 5 H, and DATA OUT 6 H).
17	CLEAR TE L	Clear out the latched TX ERROR signal.
18	TX ERROR H	A TTL output which indicates that a parity or transmission error has occurred.
19	SERIAL DATA IN H	Serial data in.
20	LOAD H	A synchronizing signal that loads data from the output buffer to the final output latch.

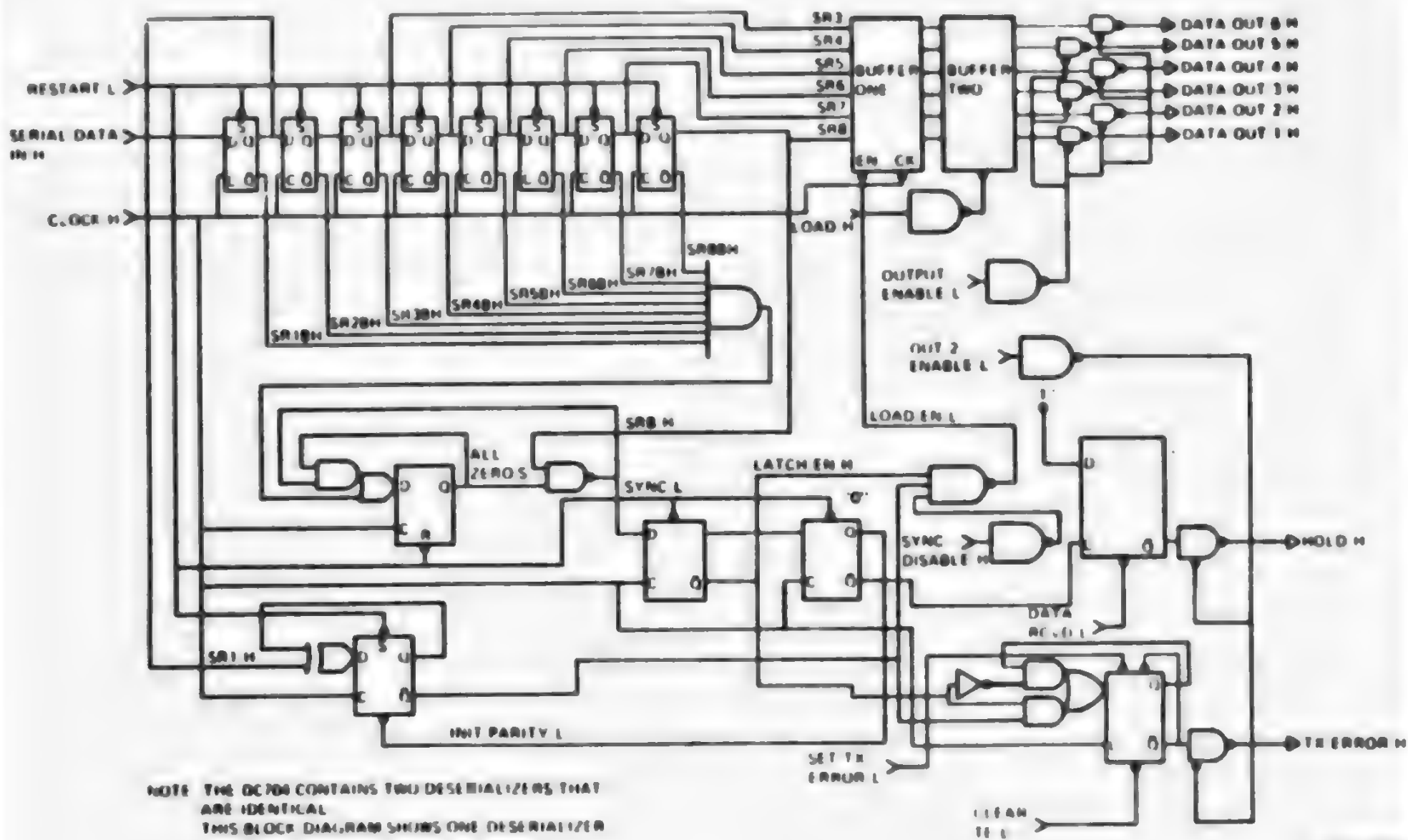
Table 1-5 (Cont.) Dual Deserializer Signal Definitions and Functions

Pin Number	Function	Description
21	DATA OUT 6 H	Parallel output data for the D6 data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted.
22	ENABLE SYNC L	When asserted, the output buffer cannot be updated.
23	DATA OUT 2 H	Parallel output data for the D2 data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted.
24	DATA OUT 1 H	Parallel output data for the D1 data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted.
25	CLK2 H	The clock signal for all the deserializer synchronous logic. The logic is positive edge triggered. This pin is tied to pin 27.
26	RESTART L	Asynchronously sets the chip to its initial conditions.
27	CLK 2 H	The clock signal for all the deserializer synchronous logic. The logic is positive edge triggered. This pin is tied to pin 25.
28	DATA OUT 3A H	Parallel output data for the D3A data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted.
29	OUTPUT 2 ENABLE A L	The input which enables tri-state outputs—RCVD H and TX ERROR H.
30	DATA OUT 5 H	Parallel output data for the D5 data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE L is asserted.
31	NTS DOS H	The non tri-state (TTL) output bit five.
32	OUTPUT ENABLE A L	The input which enables the tri-state outputs (DATA OUT 1A H, DATA OUT 2A H, DATA OUT 3A H, DATA OUT 4A H, DATA OUT 5A H, and DATA OUT 6A H).
33	DATA OUT 6A H	Parallel output data for the D6A data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted.
34	DATA OUT 2A H	Parallel output data for the D2A data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted.
35	DATA OUT 1A H	Parallel output data for the D1A data bit. Outputs are tri-state and are enabled when OUTPUT ENABLE A L is asserted.
36	V _{CC}	+5.0 volts
37	V _{CA}	Supplied from V _{CC} through a 10 ohm $\pm 5\%$, 2 watt metal film resistor.
38	SET TX ERROR A L	This input sets the TX ERROR flip flop to a high.
39	TX ERROR A H	A TTL output that indicates a parity or transmission error has occurred.

Table 1-5 (Cont.) Dual Deserializer Signal Definitions and Functions

Pin Number	Function	Description
40	ENABLE SYNC A L	When asserted, the output buffer cannot be updated
41	CLEAR TE A L	Clear out the latched TX ERROR signal
42	Unused	
43		
44	SERIAL DATA IN A H	Serial data in
45	RESTART A L	Asynchronously sets the chip to its initial buffer
46	CLK 2A H	The clock signal for all the deserializer synchronous logic. Positive edge triggered. This pin is tied to pin 48.
47	LOAD A H	A synchronizing signal that loads data from the output buffer to the final output latch.
48	CLK 2A H	The clock signal for all the deserializer synchronous logic. Positive edge triggered. This pin is tied to pin 46.

Figure 1-9 Dual Deserializer Functional Block Diagram



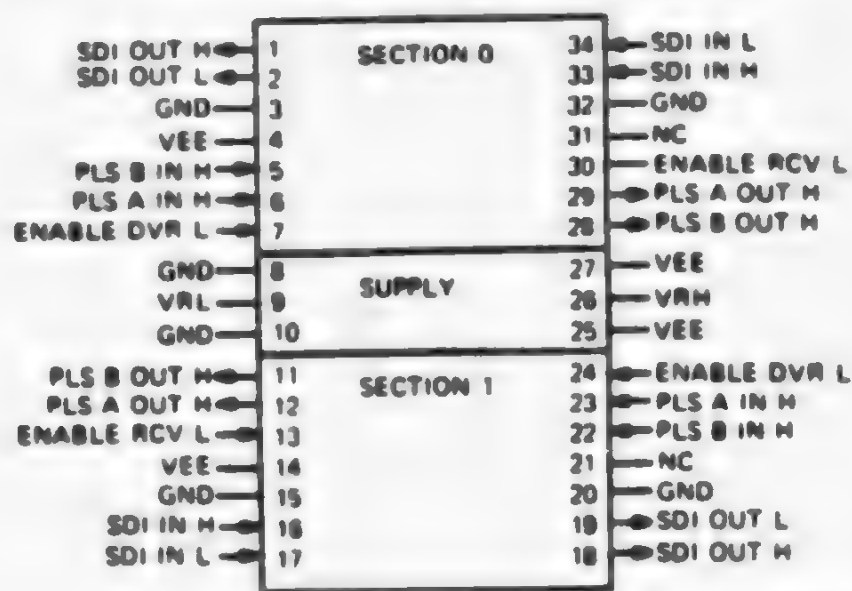
CH - 10754

1.5 SDI/STI RECEIVER/DRIVER (DC026)

The SDI/STI Receiver/Driver chip is a Digital Equipment Corporation-designed hybrid chip. This chip provides the circuit and logic necessary to perform the receiving and driving function for the SDI/STI bus at high data rates (minimum serial bit time is 40 nanoseconds). The driver receives 12 nanosecond pulses on the PLSA and PLSB inputs which have previously been encoded by the SDI/STI Encoder/Decoder chip. The driver (designed for a 93 ohm coax) converts these pulses into a three-level signal and drives the SDI/STI cable. The receiver circuitry receives the three-level signal from the cable and converts it into pulses on the PLSA and PLSB outputs. The signals are then decoded into NRZ data and clocked by the Encoder/Decoder hybrid chip. This chip uses ECL signal levels.

Figure 1-10 shows the pin layout of the dual deserializer, and Table 1-6 provides a functional description of the signals on the output pins. Figure 1-11 is a functional block diagram of the dual deserializer.

Figure 1-10 SDI/STI Receiver/Driver Pin Layout

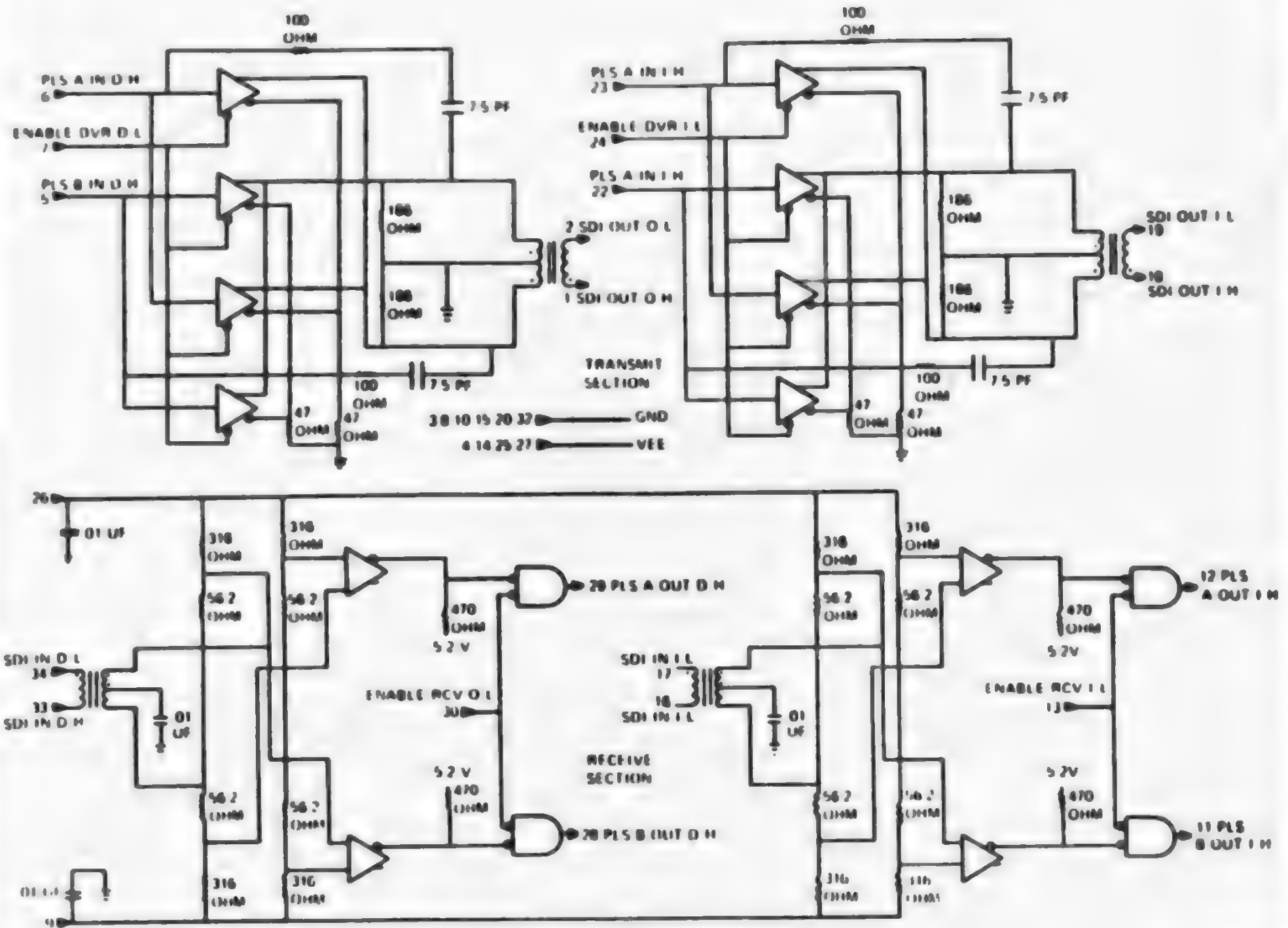


CX-1001A

Table 1-6 SDI/STI Receiver/Driver Signal Definitions and Functions

Pin Numbers	Description	Function
6 23	PLS A IN H	Input signal from the encoder PULSE A output. A high pulse on this input causes the signal at SDI OUT to pulse above the zero volt level.
5 22	PULSE B IN H	Input signal from the encoder PULSE B output. A high pulse on this input causes the signal at SDI OUT to pulse below the zero volt level.
7 24	ENABLE DVR L	Enable input for the SDI OUT H and SDI OUT L outputs maintains a zero volt level at the SDI outputs when HIGH.
2 19	SDI OUT L	The negative output connection to the SDI cable.
1 18	SDI OUT H	The positive output connection to the SDI cable.
3 8 10 15 20 32	GND	Pins 8 and 10 are major current path, pins 3, 15, 20, and 32 are additional ground paths.
4 14 25 27	V _{EE}	The -5.2 volt supply pins.
17 34	SDI IN L	The negative input connection from the SDI cable.
16 33	SDI IN H	The positive input connection from the SDI cable.
13 30	ENABLE RCV L	Enables the receive section when low.
12 29	PLS A OUT H	The output which pulses high when the signal at SDI pulses above the zero volt level. It drives PULSE A IN H of the decoder.
11 28	PLS B OUT H	The output which pulses high when the signal at SDI pulses below the zero volt level. It drives PULSE B IN H of the decoder.
26	VRH	Receive reference voltage high (+0.6 volts)
9	VRL	Receive reference voltage low (-3.3 volts)
21 31		No connection

Figure 1-11 SDI/STI Receiver/Driver Functional Block Diagram



CX-1000A

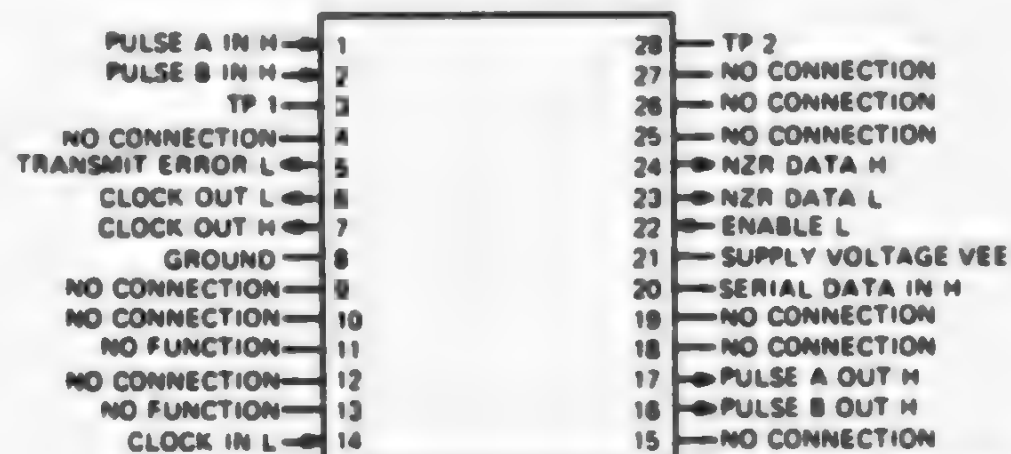
1.6 SDI/STI ENCODER/DECODER (DC027)

The SDI/STI Encoder/Decoder chip is a Digital Equipment Corporation-designed hybrid chip. This chip performs the encoding and decoding function for the SDI/STI bus.

The chip contains an encoder and a decoder. The encoder receives serial NRZ data and clock as inputs and utilizes a self-clocked encoding scheme to translate them into a series of 12-nanosecond pulses on the pulse A and pulse B outputs. These outputs drive the SDI/STI Receiver/Driver chips which drive the SDI/STI cable with the encoded signal. The decoder receives a series of pulses from the Receiver/Driver chip and performs a decoding algorithm to regenerate NRZ data and clock. This chip uses ECL signal levels.

Figure 1-12 shows the pin layout of the dual deserializer, and Table 1-7 provides a functional description of the signals on the output pins. Figure 1-13 is a functional block diagram of the dual deserializer.

Figure 1-12 SDI/STI Encoder/Decoder Pin Layout



CX-1082A

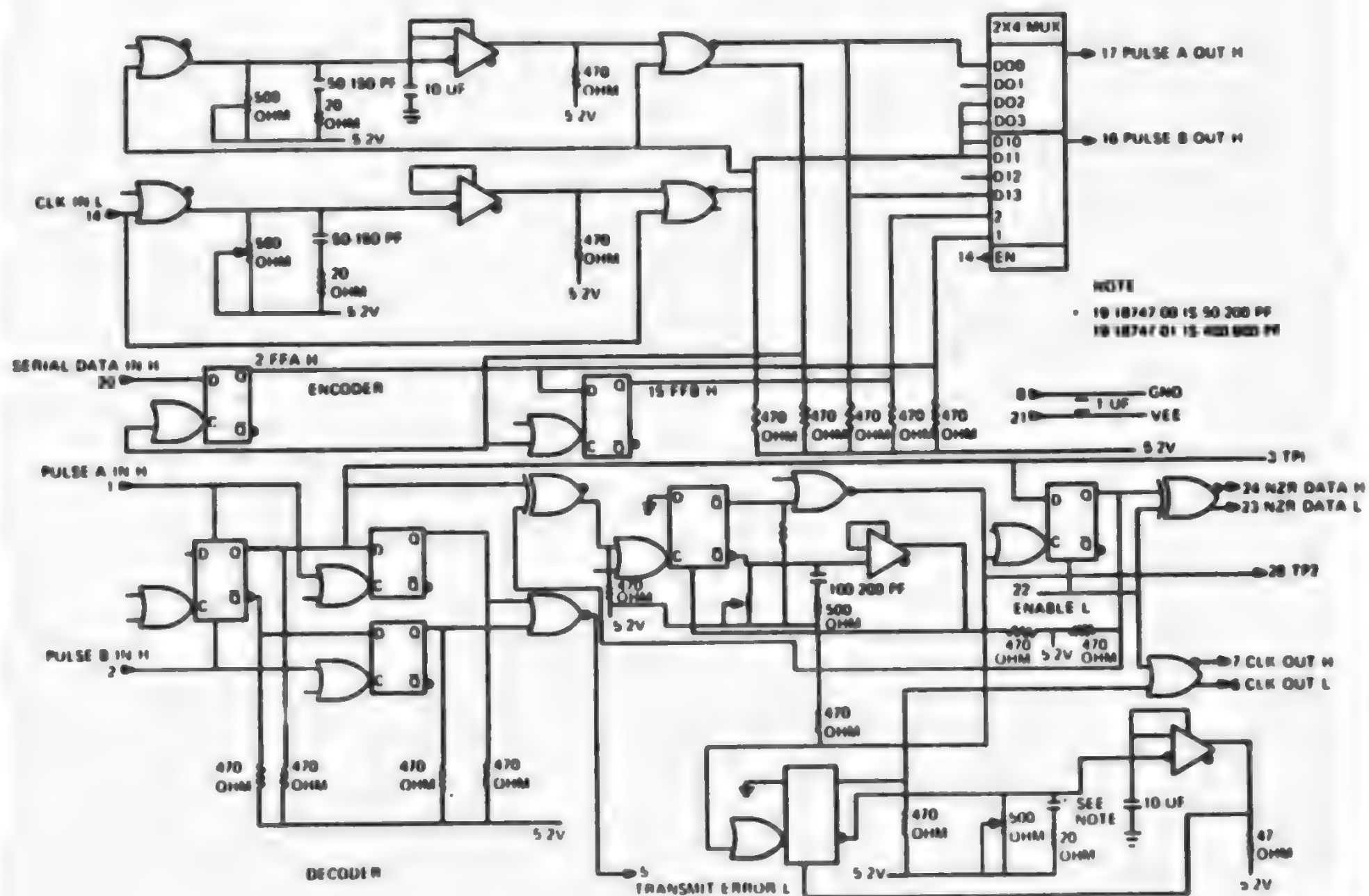
Table 1-7 SDI/STI Encoder/Decoder Signal Definitions and Functions

Pin Number	Description	Function
1	PULSE A IN H	One of the pulses (approximately 12 nanoseconds) supplied to the decoder. This signal, in conjunction with PULSE B IN H, determines the NRZ data pattern and clock. This signal is supplied from pin 29 (PLS A OUT 0 H) or pin 12 (PLS A OUT 1 H) of the SDI/STI RCV/DVR hybrid chip.
2	PULSE B IN H	One of the pulses (approximately 12 nanoseconds) supplied to the decoder. This signal, in conjunction with PULSE A IN H, determines the NRZ data pattern and clock. This signal is supplied from pin 28 (PLS B OUT 0 H) or pin 11 (PLS B OUT 1 H) of the SDI/STI RCV/DVR hybrid chip.
3	TP1	No function. Used during manufacture for resistor trimming of the decoder.
4		No connection.
5	TRANSMIT ERROR L	This output will pulse low whenever a transmission error has occurred. (For example, extra pulses or a missed pulse on either PULSE A or PULSE B inputs to the decoder.)
6	CLK OUT L	The decoded clock signal of the decoder synchronous to NRZ DATA H and NRZ DATA L.
7	CLK OUT H	
8	GND	Ground.
9		No connection.
10		
11		No function; internal connection.
12		No connection.
13		No function; internal connection.
14	CLOCK IN L	The clock input signal for the encoder; negative edge triggered. This is the timing reference for PULSE A OUT H and PULSE B OUT H.
15		No connection.
16	PULSE B OUT H	One of the 12-nanosecond pulses out of the encoder. This signal is connected to pin 5 (PLS B IN 0 H) or pin 27 (PLS B IN 1 H) of the SDI/STI RCV/DVR hybrid chip.
17	PULSE A OUT H	One of the 12-nanosecond pulses out of the encoder. The absence or presence and position of this pulse with respect to PULSE B OUT H determines the encoded data pattern. This signal is connected to pin 6 (PLS A IN 0 H) or pin 23 (PLS A IN 1 H) of the SDI/STI RCV/DVR hybrid chip.

Table 1-7 (Cont.) SDI/STI Encoder/Decoder Signal Definitions and Functions

Pin Number	Description	Function
18		No connection.
19		
20	SERIAL DATA IN H	The NRZ serial data supplied to the encoder.
21	V _{EE}	-5.2 volts.
22	ENABLE L	A high at this decoder input will cause: CLOCK OUT L (pin 6) to be in the high state. CLOCK OUT H (pin 7) to be in the low state. NRZ DATA H (pin 24) to be in the low state. NRZ DATA L (pin 23) to be in the high state. As long as this input is high, the PULSE A and PULSE B inputs have no effect on the output. When this input goes low, the clock and NRZ DATA are controlled by the clock PULSE A and PULSE inputs.
23	NRZ DATA L	The decoded NRZ data outputs which are synchronous with CLK OUT H and CLK OUT L.
24	NRZ DATA H	
25		No connection.
26		
27		
28	TP2	No function; used during manufacture for active trimming

Figure 1-13 SDI/STI Encoder/Decoder Functional Block Diagram



CX-1002A

CHAPTER 2 HSC50/70 PAL EQUATIONS

This chapter contains the PAL equations for the PALs used in the HSC50 and HSC70. This chapter is arranged by module in the following order:

- HSC50 Input/Output Control Processor Module (P.ioc)
- HSC70 Input/Output Control Processor Module (P.ioj)
- HSC70 Memory Module
- Data Channel Module (K.sdi and K.sti)

2.1 PAL Equations for PALs on the HSC50 Input/Output Control Processor Module

Figures 2-1 through 2-5 are the PAL equations for the PALs on the HSC50 Input/Output Control Processor Module.

2.2 PAL Equations for PALS on the HSC70 Input/Output Control Processor Module

Figures 2-6 through 2-13 are the PAL equations for the PALs on the HSC70 Input/Output Control Processor Module.

2.3 PAL Equations for PALS on the HSC70 Memory Module

Figures 2-14 through 2-22 are the PAL equations for the PALs on the HSC70 Memory Module.

2.4 PAL Equations for PALS on the Data Channel Module

Figure 2-23 is the PAL equation for the PAL on the Data Channel Module.

Figure 2-1 23-062J5-00

DATE: 3-APR-80

SYMBOL TABLE

01	=	1	+	2	+	20	=	VCC
02	=	2	+	3	+	19	=	/19
03	=	3	+		+	18	=	/P18_RD_VADR_REG
04	=	4	+	0	+	17	=	/P18_RD_WIND_REG
05	=	5	+	6	+	16	=	/P18_RD_VBUS_REG
06	=	6	+	2	+	15	=	/P18_RD_REPLY_0
07	=	7	+	7	+	14	=	/P18_RD_LO_EAR
08	=	8	+	5	+	13	=	/P18_RD_HI_EAR
09	=	9	+	2	+	12	=	12
10	=	10	+	2	+	11	=	11

EQUATIONS

IF [VCC] /19 = 01 + /02 + /03 + /04 + /05 + 09
;ENABLE DECODER FOR ADDRESSES 17 770 040 - 056

IF [VCC] /18_RD_VADR_REG = 01 + /02 + /03 + /04 + /05 + 09
;ENABLE ADDRESSES 17 770 000 - 016

IF [VCC] /18_RD_WIND_REG = 01 + /02 + /03 + /04 + 06 + /06
+ /07 + /08 + 09
;DECODES 17 770 020

IF [VCC] /18_RD_VBUS_REG = 01 + /02 + /03 + /04 + 06 + /06
+ /07 + 08 + 09
;DECODES 17 770 022

IF [VCC] /18_RD_REPLY_0 = 01 + /02 + /03 + 04 + 06 + 09
+ 01 + /02 + /03 + /04 + /05 + 09
+ 01 + /02 + /03 + /04 + 06 + /06 + /07 + /08 + 09
+ 01 + /02 + /03 + /04 + 06 + /06 + /07 + 08 + 09
+ 01 + /02 + /03 + /04 + 06 + /06 + 07 + /08 + 09
+ 01 + /02 + /03 + /04 + 06 + /06 + 07 + 08 + 09
;INCLUSIVE OR OF PREVIOUS FOUR TERMS AND THE FOLLOWING TWO TERMS

IF [VCC] /P18_RD_LO_EAR = 01 + /02 + /03 + /04 + 06 + /06 + 07 + /08 + 09
;DECODES 17 770 024

IF [VCC] /P18_RD_HI_EAR = 01 + /02 + /03 + /04 + 06 + /06 + 07 + 08 + 09
;DECODES 17 770 026

TITLE: E93 IO PAGE RD DECODER (LOW) 16L8
NOTE: 062J5 REV A 23-062J5-01 (P10C1)

Figure 2-2 23-063J5-00

DATE: 3-APR-80

SYMBOL TABLE

01	=	1	•	2	•	20	=	VCC
02	=	2	•	3	•	19	=	/P18_RD_RCER
03	=	3	•		•	18	=	/P18_RD_RBUF
04	=	4	•	0	•	17	=	/P18_RD_XCER
05	=	5	•	6	•	16	=	/P18_DL_SEL_1
06	=	6	•	2	•	15	=	/P18_DL_SEL_0
07	=	7	•	J	•	14	=	/P18_RD_REPLY_1
08	=	8	•	5	•	13	=	13
09	=	9	•	2	•	12	=	12
10	=	10	•	2	•	11	=	11

EQUATIONS

IF [VCC] /P18_RD_RCER = /01 • 02 • /03 • 04 • /05 • 06 • /08
• /09 • /11
• /01 • 02 • /03 • 04 • 05 • 06 • /07 • /08 • /09 • /11
;ENABLES ADDRESSES 17 777 520, -530, AND -560

IF [VCC] /P18_RD_RBUF = /01 • 02 • /03 • 04 • /05 • 06 • /08
• 09 • /11
• /01 • 02 • /03 • 04 • 05 • 06 • /07 • /08 • 09 • /11
;ENABLES ADDRESSES 17 777 522, -532, AND -562

IF [VCC] /P18_RD_XCER = /01 • 02 • /03 • 04 • /05 • 06 • 08
• /09 • /11
• /01 • 02 • /03 • 04 • 05 • 06 • /07 • 08 • /09 • /11
;ENABLES ADDRESSES 17 777 524, -534, AND -564

IF [VCC] /P18_DL_SEL_1 = /01 • 02 • /03 • 04 • 06 • /07 • /11
;ASSERTED FOR TU-1 AND TERM REGISTERS

IF [VCC] /P18_DL_SEL_0 = /01 • 02 • /03 • 04 • /05 • 06 • /07
• /11
• /01 • 02 • /03 • 04 • 05 • 06 • /07 • /11
;ASSERTED FOR TU-2 AND TERM REGISTERS

IF [VCC] /P18_RD_REPLY_1 = /01 • 02 • /03 • 04 • /05 • 06 • /11
• /01 • 02 • /03 • 04 • 05 • 06 • /07 • /11

TITLE: E94 IO PAGE RD DECODER (HIGH) 16LS
NOTE: PAT-063J5 REV A 23-063J5-1 (PIOC2)

Figure 2-3 23-064J5-00

DATE: 3-APR-80

SYMBOL TABLE

```

-----
01 = 1 * 2 * 20 = VCC
02 = 2 * 3 * 19 = /P18_LD_VADR_REG
03 = 3 * 4 * 18 = /P18_LD_VIND_REG
04 = 4 * 5 * 17 = /P18_LD_PIO_CSR
05 = 5 * 6 * 16 = /P18_LD_SWD_REG
06 = 6 * 7 * 15 = /P18_LD_KIN_REG
07 = 7 * 8 * 14 = /P18_LD_KST_REG
08 = 8 * 9 * 13 = /P18_LD_REPLY_0
09 = 9 * 10 * 12 = 12
10 = 10 * 11 * 11 = 11
-----

```

EQUATIONS

```

IF [VCC] /P18_LD_VADR_REG = 01 * /02 * /03 * /04 * /05 * 09
:ENABLES ADDRESSES 17 777 000 - 016

IF [VCC] /P18_LD_VIND_REG = 01 * /02 * /03 * /04 * /05 * /06 * /07
* 08 * 09
:ADDRESS 17 777 020

IF [VCC] /P18_LD_PIO_CSR = 01 * /02 * /03 * 04 * /05 * /06 * /07
* /08 * 09
:ADDRESS 17 777 040

IF [VCC] /P18_LD_SWD_REG = 01 * /02 * /03 * 04 * /05 * /06 * /07
* 08 * 09
:ADDRESS 17 777 042

IF [VCC] /P18_LD_KIN_REG = 01 * /02 * /03 * 04 * /05 * /06 * 07
* /08 * 09
:ADDRESS 17 777 044

IF [VCC] /P18_LD_KST_REG = 01 * /02 * /03 * 04 * /05 * /06 * 07
* 08 * 09
:ADDRESS 17 777 026

IF [VCC] /P18_LD_REPLY_0 = 01 * /02 * /03 * /04 * /05 * 09
* 01 * /02 * /03 * /04 * 05 * /06 * /07 * /08 * 09
* 01 * /02 * /03 * 04 * /05 * /06 * 09

TITLE: E103 IO PAGE LD DECODER (LOW) 16L8
NOTE: PAT-064J5 REV A 23-064J5-01 (PIOC3)

```

Figure 2-4 23-065J5-00

DATE: 3-APR-80

SYMBOL TABLE

01 = 1	2	20 = VCC
02 = 2	3	19 = /19
03 = 3	4	18 = /P18_LD_TERM_RCSR
04 = 4	0	17 = /P18_LD_TERM_XCSR
05 = 5	6	16 = /P18_LD_TERM_XBUF
06 = 6	2	15 = /P18_LD_REPLY_1
07 = 7	J	14 = 14
08 = 8	5	13 = 13
09 = 9	2	12 = 12
10 = 10	2	11 = 11

EQUATIONS

```

IF [VCC] /19 = /01 * 02 * /03 * 04 * /05 * 06 * /11
:ENABLES ADDRESSES 17 777 520 - 536

IF [VCC] /P18_LD_TERM_RCSR = /01 * 02 * /03 * 04 * 05 * 06
* /07 * 08 * /09 * /11
:ADDRESS 17 777 540

IF [VCC] /P18_LD_TERM_XCSR = /01 * 02 * /03 * 04 * 05 * 06
* /07 * 08 * /09 * /11
:ADDRESS 17 777 544

IF [VCC] /P18_LD_TERM_XBUF = /01 * 02 * /03 * 04 * 05 * 06
* /07 * 08 * 09 * /11
:ADDRESS 17 777 548

IF [VCC] /P18_LD_REPLY_1 = /01 * 02 * /03 * 04 * /05 * /11
* /01 * 02 * /03 * 04 * 05 * 06 * /07 * /11
:

```

TITLE: E113 IO PAGE LD DECODER (HIGH) 16L8
NOTE: PAT-065J5 REV A 23-065J5-01 (PIOC4)

1) REPLY IS GENERATED IF AN ATTEMPT TO WRITE AN RBUF IS MADE
HOWEVER, THE DATA IS DUMPED ON THE FLOOR (RBUF'S ARE
READ-ONLY).

For Internal Use Only
HSC80/79 PAL EQUATIONS

Figure 2-5 23-135K5-00

```

T  PAL1688
P  23-135K5-00
H  HARDWARE ENGINEER
D  22-OCT-1985

S  01 02 03 04 05 06 07 08 09 GND /11 /CORANT_7 /CORANT_6 /CORANT_5
    /CORANT_4 /CORANT_3 /CORANT_2 /CORANT_1 /CORANT_0 VCC

E  /CORANT_0 = 02 * /03 * /04 * /05 * /06 * /07 * /08 * /09
    /CORANT_1 = 03 * /04 * /05 * /06 * /07 * /08 * /09
    /CORANT_2 = 04 * /05 * /06 * /07 * /08 * /09
    /CORANT_3 = 05 * /06 * /07 * /08 * /09
    /CORANT_4 = 06 * /07 * /08 * /09
    /CORANT_5 = 07 * /08 * /09
    /CORANT_6 = 08 * /09
    /CORANT_7 = 09

```

E This PAL generates CONTROL BUS GRANTS in response to CONTROL BUS REQUESTS on the L0105 (P ioc) module of the HSC50 REQ7 is the highest priority and REQ0 is the lowest priority.

The base part number for this PAL is 23-000K5-04

Figure 2-6 23-161K3-00

T: PAL16R4
P: 23-161K3-00
H: HARDWARE ENGINEER
D: 27-OCT-84

S: STB (2) (3) 4 5 6 7 8 9 GND
(ENABLE) 12 13 /R3 /R2 /R1 /R0 18 19 VCC

S: /R0 = 12 • /6 • /8 • /4 • /13 • /19
• 7 • /8 • /4 • /13 • /19
• 9 • /4 • /13 • /19
• 5 • /13 • /19
• 18 • /19

/R1 = 12 • /6 • /7 • /9 • /18 • /19
• 8 • /9 • /18 • /19
• 4 • /18 • /19
• 5 • /18 • /19
• 13 • /18 • /19

/R2 = 12 • /9 • /4 • /5
• 6 • /9 • /4 • /5
• 7 • /9 • /4 • /5
• 8 • /9 • /4 • /5
• 13
• 18
• 19

/R3 = 6 • /13 • /18 • /19
• 7 • /13 • /18 • /19
• 8 • /13 • /18 • /19
• 9 • /13 • /18 • /19
• 4 • /13 • /18 • /19
• 5 • /13 • /18 • /19

(Continued on next page)

Figure 2-6 (Cont.) 23-161K3-00

E: END OF EQUATIONS

E133. INTERRUPT PRIORITY ENCODER

This PAL generates a 4-bit code corresponding to the highest priority active interrupt of the ten possible interrupts on the module. Synchronization of the interrupts is provided external to the PAL.

Valid codes are defined in the following table:

	R3	R2	R1	R0
CONTROL BUS 7	H	L	H	H
CONTROL BUS 6	H	L	H	L
CONTROL BUS 5	H	L	L	H
TERMINAL RCVR	L	H	L	L
TERMINAL XMITR	L	H	L	H
AUX1 RCVR	L	H	H	L
AUX1 XMITR	L	L	L	H
AUX2 RCVR	L	L	H	L
AUX2 XMITR	L	L	H	H
CONTROL BUS 4	H	L	L	L
NO INTERRUPTS ACTIVE	H	H	H	H

NOTE: Input terms (2) and (3) are not presently used, but could be used for two additional interrupt inputs. Interrupts are encoded asynchronously and appear at the registered outputs (R3.0) after the clock occurs.

This PAL is used on the L0111 (P.10j) module of the NRC70

The base part number for this PAL is 23-000K3-03

Figure 2-7 23-258J5-00

```

T:  PAL16L8
P:  23-258J5-00
S:  HARDWARE ENGINEER
D:  22-MAR-83

S:  01 02 03 04 05 06 07 08 09 GND
    11 12 13 14 15 16 17 18 19 VCC

B:  IF [VCC] /17 = 03 = 04 = 05 = 06 = 07 = 08 = 09 = 11 = 13 = 14
    = 01
    = /02 = 03
    IF [VCC] /16 = /01 = /03 = 04 = 05 = 06 = 07 = 08 = 09 = 11 = 13 = 14
    IF [VCC] /15 = /01 = 02 = /04 = 05 = 06 = 07 = 08 = 09 = 11 = 13 = 14

E:  END OF EQUATIONS

```

TITLE: Low Order Bus Arbitrator

This PAL is used for both the CONTROL BUS and the DATA BUS ARBITRATORS, in conjunction with 23-258J5-00. It asserts a GRANT for REQ1 if it is asserted, or for REQ0 if it is asserted and REQ1 is not asserted, assuming no higher priority request is asserted (REQ0 - REQ2) and the enable inputs (pins 1 and 2) are in the proper state. Pin 17 generates a REQUEST ACTIVE output which is asserted whenever any of the ten GRANTs is asserted.

This PAL is used on the L0111 (P.10j) module of the HSC70.

The base part number for this PAL is 23-000J5-03.

For Internal Use Only
HSC80/70 PAL EQUATIONS

Figure 2-8 23-259J5-00

```

T:  PAL16LE
P:  23-259J5-00
B:  HARDWARE ENGINEER
D:  23-MAR-83

S:  01 02 03 04 05 06 07 08 09 GND
    11 12 13 14 15 16 17 18 19 VCC

B:  IF [VCC] /19 = /01 * 02 * /03 * 04 * 05 * 06 * 07 * 08 * 09 * 11
    IF [VCC] /18 = /01 * 02 * /05 * 06 * 07 * 08 * 09 * 11
    IF [VCC] /17 = /01 * 02 * /06 * 07 * 08 * 09 * 11
    IF [VCC] /16 = /01 * 02 * /07 * 08 * 09 * 11
    IF [VCC] /15 = /01 * 02 * /08 * 09 * 11
    IF [VCC] /14 = /01 * 02 * /09 * 11
    IF [VCC] /13 = /01 * 02 * /11
    IF [VCC] /12 = /01 * 02 * /04 * 05 * 06 * 07 * 08 * 09 * 11

E:  END OF EQUATIONS

```

TITLE: High Order Bus Arbitrator

This PAL is used for both the CONTROL BUS and the DATA BUS ARBITRATORS, in conjunction with 23-259J5-00. It asserts a GRANT for the highest priority asserted REQ (REQ9 - REQ2) if the enable pins (pins 1 and 2) are in the correct state.

This PAL is used on the L0111 (P 10j) module of the HSC70.

The base part number for this PAL is 23-000J5-03.

Figure 2-9 23-260J5-00

```

T: PAL16LH
P: 23-260J5-00
H: HARDWARE ENGINEER
D: 3-APR-80

S: 01 02 03 04 05 06 07 08 09 GND
    11 12 /P18_RD_HI_EAR /P18_RD_LO_EAR /P18_RD_REPLY_0 /P18_RD_WBUS_REG
    /P18_RD_WIND_REG /P18_RD_WADR_REG /19 VCC

B: IF [VCC] /19 = 01 * /02 * /03 * 04 * /05 * 09
    ;ENABLES DECODER FOR ADDRESSES 17 770 040 - 056

    IF [VCC] /P18_RD_WADR_REG = 01 * /02 * /03 * /04 * /05 * 09
    ;ENABLES ADDRESSES 17 770 000 - 016

    IF [VCC] /P18_RD_WIND_REG = 01 * /02 * /03 * /04 * 05 * /06
    * /07 * /08 * 09
    ;DECODES 17 770 020

    IF [VCC] /P18_RD_WBUS_REG = 01 * /02 * /03 * /04 * 05 * /06
    * /07 * 08 * 09
    ;DECODES 17 770 022

    IF [VCC] /P18_RD_REPLY_0 = 01 * /02 * /03 * 04 * /05 * 09
    * 01 * /02 * /03 * /04 * /05 * 09
    * 01 * /02 * /03 * /04 * 05 * /06 * /07 * /08 * 09
    * 01 * /02 * /03 * /04 * 05 * /06 * /07 * 08 * 09
    * 01 * /02 * /03 * /04 * 05 * /06 * 07 * /08 * 09
    ;INCLUSIVE OR OF PREVIOUS FOUR TERMS AND THE FOLLOWING TWO TERMS

    IF [VCC] /P18_RD_LO_EAR = 01 * /02 * /03 * /04 * 05 * /06 * 07 * /08
    * 09
    ;DECODES 17 770 024

    IF [VCC] /P18_RD_HI_EAR = 01 * /02 * /03 * /04 * 05 * /06 * 07 * 08 * 09
    ;DECODES 17 770 026

E: END OF EQUATIONS

TITLE E91. 10 PAGE RD DECODER

This PAL is used on the L0111 (P.10J) module of the HSC70

The base part number for this PAL is 23-000J5-04.

```


For Internal Use Only
HSC50/70 PAL EQUATIONS

Figure 2-10 23-261J5-00

```

T: PAL16L8
P: 23-261J5-00
H: TOM FAVA
D: 3-APR-80

S: 01 02 03 04 05 06 07 08 09 GND
    11 12 /P18_LD_REPLY_0 /P18_LD_KST_REG
    /P18_LD_KIN_REG /P18_LD_SVD_REG /P18_LD_PIO_CSR
    /P18_LD_WIND_REG /P18_LD_WADR_REG VCC

B: IF [VCC] /P18_LD_WADR_REG = 01 * /02 * /03 * /04 * /05 * 09
    ;ENABLES ADDRESSES 17 770 000 - 016
    IF[VCC] /P18_LD_WIND_REG = 01 * /02 * /03 * /04 * 05 * /06 * /07
    * /08 * 09
    ;ADDRESS 17 770 020
    IF [VCC] /P18_LD_PIO_CSR = 01 * /02 * /03 * 04 * /05 * /06 * /07
    * /08 * 09
    ;ADDRESS 17 770 040
    IF [VCC] /P18_LD_SVD_REG = 01 * /02 * /03 * 04 * /05 * /06 * /07
    * 08 * 09
    ;ADDRESS 17 770 042
    IF [VCC] /P18_LD_KIN_REG = 01 * /02 * /03 * 04 * /05 * /06 * 07
    * /08 * 09
    ;ADDRESS 17 770 044
    IF [VCC] /P18_LD_KST_REG = 01 * /02 * /03 * 04 * /05 * /06 * 07
    * 08 * 09
    ;ADDRESS 17 770 046
    IF [VCC] /P18_LD_REPLY_0 = 01 * /02 * /03 * /04 * /05 * 09
    * 01 * /02 * /03 * /04 * 05 * /06 * /07 * /08 * 09
    * 01 * /02 * /03 * 04 * /05 * /06 * 09

E: END OF EQUATIONS

IO PAGE WRITE DECODE

This PAL is used on the L0111 (P.10j) module of the HSC70.
The base part number for this PAL is 23-000J5-04

```

Figure 2-11 23-262J5-00

```

T: PAL16L8
P: 23-262J5-00
E: HARDWARE ENGINEER
D: 23-FEB-86

S: NO R1 R2 R3 (5) (6) (7) (8) (9) GND
    TIAK IN_BUS4 /INTR_EN TIAKO /EN_DL_CLR IN_BUS7 IN_BUS6 IN_BUS5 IN_BUS3
    VCC

B: IF [INTR_EN] /IN_BUS3 = R2
    * /R1

    IF [INTR_EN] /IN_BUS4 = /R3 * /R2
    * R2 * R1

    IF [INTR_EN] /IN_BUS5 = R3
    * /R2
    * R1

    IF [INTR_EN] /IN_BUS6 = R2 * /R1
    IF [INTR_EN] /IN_BUS7 = R3
    * R2 * /R1

    IF [VCC] /INTR_EN = /R2 * TIAK
    * /R3 * TIAK

    IF [VCC] /TIAKO = /R0
    * /R1
    * /R2
    * /R3
    * /TIAK

    IF [VCC] /EN_DL_CLR = /R3 * INTR_EN

```

(Continued on next page)

Figure 2-11 (Cont.) 23-262J5-00

E: END OF EQUATIONS

E123, INTERRUPT VECTOR GENERATOR

This PAL is used on the L0111 (P.1a) module of the HSC70.

The base part number for this PAL is 23-000J5-04.

This PAL generates bits 07:03 of the interrupt vector for interrupts generated on this module; vector bits 02:00 are generated externally to the PAL. The interrupts, defined by inputs R3:0, and vectors are shown below:

	R 3:0	VECTOR
CONTROL BUS 7	HLHN	134
CONTROL BUS 6	HLHL	130
CONTROL BUS 5	HLLN	124
TERMINAL RCVR	LHLL	080
TERMINAL XMTR	LHLN	084
AUX1 RCVR	LHNL	300
AUX1 XMTR	LLLN	304
AUX2 RCVR	LLHL	310
AUX2 XMTR	LLHN	314
CONTROL BUS 4	NLLL	120

Output INTR_EN is used to enable other parts of the interrupt logic during an interrupt cycle.

Output TIAKO propagates the Interrupt Acknowledge signal to the Program Memory Bus in the event a Level 4 interrupt is generated external to the module (ie, by the RX33 controller).

Output EN_DL_CLR is used to enable a clear signal to the DLART interrupt output currently being serviced.

Figure 2-12 23-263J5-00

```

T: PAL16L8
P: 23-263J5-00
H: HARDWARE ENGINEER
D: 30-OCT-85

S: R3 R2 R1 R0 (5) RIRQ4_IR /TINIT /IR_EN (9) GND
(11) IRQ_5 /CLR_C_IRQ4 /CLR_C_IRQ5 /CLR_C_IRQ6 /CLR_C_IRQ7 IRQ_7 IRQ_6
IRQ_4 VCC

B: IF [VCC] /IRQ_4 = /R3 * /R2 * /R1 * /R0
      = /R3 * R2 * R1 * R0
      * R3 * R2 * /R0
      * R3 * /R2 * R1
      * /RIRQ4_IR * R3 * R1
      * R3 * /R1 * R0

      IF [VCC] /IRQ_5 = /R3
      * R2
      * R1
      * /R0

      IF [VCC] /IRQ_6 = /R3
      * R2
      * /R1
      * R0

      IF [VCC] /IRQ_7 = /R3
      * R2
      * /R1
      * /R0

      IF [VCC] /CLR_C_IRQ4 = R3 * /R2 * /R1 * /R0 * IR_EN * TINIT
      IF [VCC] /CLR_C_IRQ5 = R3 * /R2 * /R1 * R0 * IR_EN * TINIT
      IF [VCC] /CLR_C_IRQ6 = R3 * /R2 * R1 * /R0 * IR_EN * TINIT
      IF [VCC] /CLR_C_IRQ7 = R3 * /R2 * R1 * R0 * IR_EN * TINIT

E: END OF EQUATIONS

```

R125. INTERRUPT DECODER

This PAL is used on the LO111 (P.10j) module of the HSC70.

The base part number for this PAL is 23-000J5-04.

This PAL generates the four INTERRUPT LEVEL 7:4 inputs to the J-11 chip based on the R3 0 inputs and the Program Memory Bus IRQ4 input. It also generates a clear signal to the flipflop holding the Control Bus Interrupt currently being serviced.

For Internal Use Only
HSC30/70 PAL EQUATIONS

Figure 2-13 23-264J5-00

```

T:  PAL16LH
P:  23-264J5-00
E:  HARDWARE ENGINEER
D:  02-JAN-85

S:  1 2 3 4 5 6 7 8 9 GND
    11 /12 /TEST /14 /15 /16 /17 /18 /19 VCC

B:  IF [VCC] 12 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * 11
    * 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8 * 11

    IF [VCC] 14 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * 8
    IF [VCC] 15 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * /8
    IF [VCC] 16 = 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8
    IF [VCC] 17 = 1 * /2 * 3 * 4 * /TEST
    * 1 * 2 * /4 * /TEST
    * 1 * 2 * /3 * 4 * /TEST

    IF [VCC] 18 = 1 * 2 * 3 * 4 * 5 * /6 * 7 * 9
    * 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8 * 9

    IF [VCC] 19 = 1 * /2 * 3 * 4 * 9 * /TEST
    * 1 * 2 * /4 * 9 * /TEST
    * 1 * 2 * /3 * 4 * 9 * /TEST
    * 1 * 2 * 3 * 4 * 5 * /6 * 7
    * 1 * 2 * 3 * 4 * 5 * 6 * 7 * /8

E:  END OF EQUATION

```

E186. Bibus Address Decoder

This PAL is used on the LO111 (P.10j) module of the HSC70.

The base part number for this PAL is 23-000J5-04

This PAL generates RD and WRT enables for those elements on the Bibus as well as for the Bibus transceivers. These elements include the Bootstrap PROM and the DLARTs for the Console Terminal, and the AUX1 and AUX2 serial ports. A TEST input is provided to disable the bootstrap PROMs and allow the module tester to supply a specialized bootstrap program for testing the module.

Figure 2-14 23-007K7-00

```

T: PAL20X8
P: 23-007K7-00
H: HARDWARE ENGINEER
D: 17-DEC-86

S: CLK IO D0 D1 D2 D3 D4 D5 D6 D7 I1 GND
   /OC /MAX Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 /CI VCC

B: /Q0 := /I1-/IO          :CLEAR LSB
    * IO-/Q0              :COUNT/HOLD
    : I1-/IO-/D0          :LOAD D0 (LSB)
    * I1 IO CI            :COUNT

    /Q1 := /I1-/IO          :CLEAR
    * IO-/Q1              :COUNT/HOLD
    : I1-/IO-/D1          :LOAD D1
    * I1 IO CI-Q0         :COUNT

    /Q2 := /I1-/IO          :CLEAR
    * IO-/Q2              :COUNT/HOLD
    : I1-/IO-/D2          :LOAD D2
    * I1 IO CI-Q0-Q1      :COUNT

    /Q3 := /I1-/IO          :CLEAR
    * IO-/Q3              :COUNT/HOLD
    : I1-/IO-/D3          :LOAD D3
    * I1 IO CI-Q0-Q1-Q2   :COUNT

    /Q4 := /I1-/IO          :CLEAR
    * IO-/Q4              :COUNT/HOLD
    : I1-/IO-/D4          :LOAD D4
    * I1 IO CI-Q0-Q1-Q2-Q3 :COUNT

    /Q5 := /I1-/IO          :CLEAR
    * IO-/Q5              :COUNT/HOLD
    : I1-/IO-/D5          :LOAD D5
    * I1 IO CI-Q0-Q1-Q2-Q3-Q4 :COUNT

    /Q6 := /I1-/IO          :CLEAR
    * IO-/Q6              :COUNT/HOLD
    : I1-/IO-/D6          :LOAD D6
    * I1 IO CI-Q0-Q1-Q2-Q3-Q4-Q5 :COUNT

    /Q7 := /I1-/IO          :CLEAR MSB
    * IO-/Q7              :COUNT/HOLD
    : I1-/IO-/D7          :LOAD D7 (MSB)
    * I1 IO CI-Q0-Q1-Q2-Q3-Q4-Q5-Q6 :COUNT

IF [VCC] MAX = Q0-Q1-Q2-Q3-Q4-Q5-Q6-Q7 :MAX COUNT

```

(Continued on next page)

Figure 2-14 (Cont.) 23-007K7-00

E E137 MABO (Memory Address Register 0) PAL

THIS IS AN 8-BIT SYNCHRONOUS COUNTER WITH PARALLEL LOAD, CLEAR, AND HOLD CAPABILITY. THE LOAD OPERATION LOADS THE INPUTS (D7-D0) INTO THE OUTPUT REGISTER (Q7-Q0). THE CLEAR OPERATION RESETS THE OUTPUT REGISTER TO ALL LOWS. THE HOLD OPERATION HOLDS THE PREVIOUS VALUE REGARDLESS OF CLOCK TRANSITIONS. THE INCREMENT OPERATION ADDS ONE TO THE OUTPUT REGISTER WHEN THE CARRY-IN IS TRUE (/CI=L), OTHERWISE THE OPERATION IS A HOLD. MAX IS TRUE (/MAX=L) WHEN THE OUTPUT REGISTER (Q7-Q0) IS ALL HIGH.

OPERATIONS TABLE

CLK	I1	I0	/CI	D7-D0	Q7-Q0	OPERATION
X	X	X	X	X	Z	HI-Z
C	L	L	X	X	L	CLEAR
C	L	H	X	X	Q	HOLD
C	H	L	X	D	D	LOAD
C	H	H	H	X	Q	HOLD
C	H	H	L	X	Q PLUS 1	INCREMENT

This PAL is used on the L0117 (N.std2) module of the MBC70.
The base part number for this PAL is 23-000K7-01.

Figure 2-15 23-008K7-00

```

T  PAL20AH
P  23-008K7-00
H  HARDWARE ENGINEER
D  17-DEC-86

S  CLK IO D0 D1 D2 D3 D4 D5 D6 D7 I1 GND
   /OC /MAX Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 CI VCC

B  /Q0 = /I1-/I0          ;CLEAR LSB
   .      IO-/Q0          ;COUNT/HOLD
   .      I1-/IO-/D0      ;LOAD D0 (LSB)
   .      I1 IO CI        ;COUNT

   /Q1 = /I1-/I0          ;CLEAR
   .      IO-/Q1          ;COUNT/HOLD
   .      I1-/IO-/D1      ;LOAD D1
   .      I1 IO CI Q0     ;COUNT

   /Q2 = /I1-/I0          ;CLEAR
   .      IO-/Q2          ;COUNT/HOLD
   .      I1-/IO-/D2      ;LOAD D2
   .      I1 IO CI Q0 Q1  ;COUNT

   /Q3 = /I1-/I0          ;CLEAR
   .      IO-/Q3          ;COUNT/HOLD
   .      I1-/IO-/D3      ;LOAD D3
   .      I1 IO CI Q0 Q1 Q2 ;COUNT

   /Q4 = /I1-/I0          ;CLEAR
   .      IO-/Q4          ;COUNT/HOLD
   .      I1-/IO-/D4      ;LOAD D4
   .      I1 IO CI Q0 Q1 Q2 Q3 ;COUNT

   /Q5 = /I1-/I0          ;CLEAR
   .      IO-/Q5          ;COUNT/HOLD
   .      I1-/IO-/D5      ;LOAD D5
   .      I1 IO CI Q0 Q1 Q2 Q3 Q4 ;COUNT

   /Q6 = /I1-/I0          ;CLEAR
   .      IO-/Q6          ;COUNT/HOLD
   .      I1-/IO-/D6      ;LOAD D6
   .      I1 IO CI Q0 Q1 Q2 Q3 Q4 Q5 ;COUNT

   /Q7 = /I1-/I0          ;CLEAR MSB
   .      IO-/Q7          ;COUNT/HOLD
   .      I1-/IO-/D7      ;LOAD D7 (MSB)
   .      I1 IO CI Q0 Q1 Q2 Q3 Q4 Q5 Q6 ;COUNT

IF [VCC] MAX = Q0-Q1-Q2-Q3-Q4-Q5-Q6-Q7 ;MAX COUNT

```

(Continued on next page)

Figure 2-15 (Cont.) 23-008K7-00

E: E143 NAR1 (Memory Address Register 1) PAL

This is an 8-bit synchronous counter with parallel load, clear, and hold capability. The load operation loads the inputs (D7-D0) into the output register (Q7-Q0). The clear operation resets the output register to all lows. the hold operation holds the previous value regardless of clock transitions. the increment operation adds one to the output register when the carry-in is true (CI=H), otherwise the operation is a hold. Max is true (/MAX=L) when the output register (Q7-Q0) is all highs.

Operations Table:

CLK	I1	I0	CI	D7-D0	Q7-Q0	OPERATION
X	X	X	X	X	Z	HI-Z
C	L	L	X	X	L	CLEAR
C	L	H	X	X	Q	HOLD
C	H	L	X	D	D	LOAD
C	H	H	L	X	Q	HOLD
C	H	H	H	X	Q PLUS 1	INCREMENT

This PAL is used on the L0117 (N std2) module of the HSC70.

The base part number for this PAL is 23-008K7-01.

Figure 2-16 23-009K7-00

```

T: PAL20X8
P: 23-009K7-00
R: HARDWARE ENGINEER
D: 18-DEC-85

S: CLK IO D0 D1 D2 D3 D4 D5 D6 D7 I1 GND
   /OC CIN Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0 /ENB VCC

B: /Q0 := /I1-/IO          :CLEAR LSR
   * IO-/Q0                :COUNT/HOLD
   : I1-/IO-/D0            :LOAD D0 (LSB)
   * I1 IO ENB-CIN         :COUNT

/Q1 := /I1-/IO          :CLEAR
   * IO-/Q1                :COUNT/HOLD
   : I1-/IO-/D1            :LOAD D1
   * I1 IO ENB-CIN-Q0      :COUNT

/Q2 := /I1-/IO          :CLEAR
   * IO-/Q2                :COUNT/HOLD
   : I1-/IO-/D2            :LOAD D2
   * I1 IO ENB-CIN-Q0-Q1   :COUNT

/Q3 := /I1-/IO          :CLEAR
   * IO-/Q3                :COUNT/HOLD
   : I1-/IO-/D3            :LOAD D3
   * I1 IO ENB-CIN-Q0-Q1-Q2 :COUNT

/Q4 := /I1-/IO          :CLEAR
   * IO-/Q4                :COUNT/HOLD
   : I1-/IO-/D4            :LOAD D4
   * I1 IO ENB-CIN-Q0-Q1-Q2-Q3 :COUNT

/Q5 := /I1-/IO          :CLEAR
   * IO-/Q5                :COUNT/HOLD
   : I1-/IO-/D5            :LOAD D5
   * I1 IO ENB-CIN-Q0-Q1-Q2-Q3-Q4 :COUNT

/Q6 := /I1-/IO          :CLEAR
   * IO-/Q6                :COUNT/HOLD
   : I1-/IO-/D6            :LOAD D6

/Q7 := /I1-/IO          :CLEAR MSB
   * IO-/Q7                :COUNT/HOLD
   : I1-/IO-/D7            :LOAD D7 (MSB)

```

(Continued on next page)

Figure 2-16 (Cont.) 23-009K7-00

E: E150 NAR2 (Memory Address Register 2) PAL

This is a 6-bit synchronous counter with parallel load, clear, and hold capability. The load operation loads the inputs (D7-D0) into the output register (Q7-Q0). The clear operation resets the output register to all lows. The hold operation holds the previous value regardless of clock transitions. The increment operation adds one to the output register when the enable is true (/ENB=L) and carry-in is true (CIN=M). Otherwise the operation is a hold. Bits 6 & 7 are used only as register bits and do not increment.

Operations Table.

/OC	CLK	I1	I0	/ENB	CIN	D7-D0	Q5-Q0	Q7-Q6	OPERATION
H	X	X	X	X	X	X	Z	Z	HI-Z
L	C	L	L	X	X	X	L	L	CLEAR
L	C	L	H	X	X	X	Q	Q	HOLD
L	C	H	L	X	X	D	D	D	LOAD
L	C	H	H	H	X	X	Q	Q	HOLD
L	C	H	H	X	L	X	Q	Q	HOLD
L	C	H	H	L	H	X	Q PLUS 1	Q	INCREMENT/HOLD

This PAL is used on the L0117 (M std2) module of the HSC70

The base part number for this PAL is 23-000K7-01.

Figure 2-17 23-127K5-00

T: PAL16R8
P: 23-127K5-00
D: 28-AUG-85

S: PAL_CLK /BINIT /REG_SEL_0 /REG_SEL_2 /REG_SEL_4 /REG_SEL_6
/INVD SYNC_OUT_CYC PIN9 GND
/TS_OE /REG_RD /RD_MAR_2 /RD_MAR_1 /RD_MAR_0
/MAR_LOAD_DONE /LD_MAR_2 /LD_MAR_1 /LD_MAR_0 VCC

B: LD_MAR_0 = REG_SEL_2 * SYNC_OUT_CYC * /MAR_LOAD_DONE * /BINIT
* BINIT * /MAR_LOAD_DONE
LD_MAR_1 = REG_SEL_4 * SYNC_OUT_CYC * /MAR_LOAD_DONE * /BINIT
* BINIT * /MAR_LOAD_DONE
LD_MAR_2 = REG_SEL_6 * SYNC_OUT_CYC * /MAR_LOAD_DONE * /BINIT
* BINIT * /MAR_LOAD_DONE
MAR_LOAD_DONE = REG_SEL_2 * SYNC_OUT_CYC * /BINIT
* REG_SEL_4 * SYNC_OUT_CYC * /BINIT
* REG_SEL_6 * SYNC_OUT_CYC * /BINIT
* MAR_LOAD_DONE * SYNC_OUT_CYC * /BINIT
* BINIT
RD_MAR_0 = REG_SEL_2 * INVD * /BINIT
RD_MAR_1 = REG_SEL_4 * INVD * /BINIT
RD_MAR_2 = REG_SEL_6 * INVD * /BINIT
REG_RD = REG_SEL_0 * INVD
* REG_SEL_2 * INVD
* REG_SEL_4 * INVD
* REG_SEL_6 * INVD

E: E120, REGCTL (Register Control) PAL for RX33 Controller

This PAL generates signals which control the reading and writing of the floppy registers, primarily the MAR, that are external to the FDC chip. All outputs are asserted low.

BINIT will cause all LD MAR x outputs to become asserted in order to clear the MAR, (especially the Module OK bit), on power-up

Figure 2-16 23-128K5-00

```

T: PAL16R8
P: 23-128K5-00
D: 28-AUG-85

S: PAL_CLK /RESET /REG_SEL_0 /REG_SEL_2 /REG_SEL_4 /REG_SEL_6
  /INVD SYNC_OUT_CYC /DISK_RW GND
  /TS_0E /RX_LOAD_DONE RX_SEL_0 /RD_CSR /LD_CSR /RX_SLAVE_VR
  /RX_SLAVE_RD RX_SEL_1 PIN19 VCC

B: /RX_SEL_0 = /REG_SEL_2 = /REG_SEL_6 = /DISK_RW
  /RX_SEL_1 = /REG_SEL_4 = /REG_SEL_6 = /DISK_RW

RX_SLAVE_RD = REG_SEL_0 * INVD * /RESET
  * REG_SEL_2 * INVD * /RESET
  * REG_SEL_4 * INVD * /RESET
  * REG_SEL_6 * INVD * /RESET

RX_SLAVE_VR = REG_SEL_0 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
  * REG_SEL_2 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
  * REG_SEL_4 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET
  * REG_SEL_6 * SYNC_OUT_CYC * /RX_LOAD_DONE * /RESET

RX_LOAD_DONE = REG_SEL_0 * SYNC_OUT_CYC
  * REG_SEL_2 * SYNC_OUT_CYC
  * REG_SEL_4 * SYNC_OUT_CYC
  * REG_SEL_6 * SYNC_OUT_CYC
  * RX_LOAD_DONE * SYNC_OUT_CYC

LD_CSR = REG_SEL_0 * SYNC_OUT_CYC * /RX_LOAD_DONE
RD_CSR = REG_SEL_0 * INVD * /RESET

```

E: E121. RXCTL (RX Control) PAL for RX33 Controller

This PAL generates signals which control the reading and writing of the floppy registers inside the FDC chip. All outputs are asserted low except RX_SEL_0,1 which are asserted high.

RX_SEL_0 and RX_SEL_1 are both asserted whenever DISK_RW is asserted since only the RX DBUFF is used during the actual read/write transfers to the floppy drive.

This PAL also controls the reading and writing of the CSR hi byte which is external to the FDC chip.

Figure 2-19 23-131K5-00

T: PAL16R8
P: 23-131K5-00
D: 28-AUG-85

S: PAL_CLK BYTE_REQ PIN3 TDIN /R_RPLY /DISK_WRITE MASTER /DMA_TEST PIN9
GND /TS_0E /END_RBUF /RBUF_EMPTY /DMA_RD_REQ /END_RBUF_LB /END_RBUF_HB
/LD_RBUF /RX_MASTER_WR /STOP_DMA_WR_TEST VCC

H: RX_MASTER_WR = DISK_WRITE • LD_RBUF • /RX_MASTER_WR • RBUF_EMPTY
• DISK_WRITE • BYTE_REQ • /RX_MASTER_WR • /RBUF_EMPTY
• /DMA_TEST

LD_RBUF = DISK_WRITE • MASTER • TDIN • R_RPLY • RBUF_EMPTY • /LD_RBUF

END_RBUF_LB = DISK_WRITE • LD_RBUF • /END_RBUF_LB
• DISK_WRITE • END_RBUF_LB • RX_MASTER_WR

END_RBUF_HB = DISK_WRITE • BYTE_REQ • /RBUF_EMPTY • /END_RBUF_HB
• /END_RBUF_LB • /DMA_TEST
• DISK_WRITE • END_RBUF_HB • RX_MASTER_WR

DMA_RD_REQ = DISK_WRITE • BYTE_REQ • RBUF_EMPTY • /DMA_RD_REQ
• /MASTER • /END_RBUF_HB • /DMA_TEST
• DISK_WRITE • DMA_RD_REQ • /MASTER
• DISK_WRITE • /STOP_DMA_WR_TEST • RBUF_EMPTY • /DMA_RD_REQ
• /MASTER • /END_RBUF_HB • DMA_TEST

RBUF_EMPTY = /DISK_WRITE
• DISK_WRITE • /LD_RBUF • RBUF_EMPTY
• DISK_WRITE • /RBUF_EMPTY • /RX_MASTER_WR • END_RBUF_HB
• /DMA_TEST
• DISK_WRITE • /RBUF_EMPTY • DMA_TEST • END_RBUF_LB
• /RX_MASTER_WR

END_RBUF = DISK_WRITE • LD_RBUF • /END_RBUF
• DISK_WRITE • BYTE_REQ • /RBUF_EMPTY • /END_RBUF
• /END_RBUF_LB • /DMA_TEST
• DISK_WRITE • END_RBUF • RX_MASTER_WR

STOP_DMA_WR_TEST = DISK_WRITE • /DMA_TEST
• STOP_DMA_WR_TEST • DISK_WRITE
• DMA_TEST • MASTER

E: E136, DISKWR (Disk Write) Sequencer PAL for RX33 Controller

This PAL generates signals steer the data path during writes to the floppy disk. DMA test mode allows data to be transferred from memory to the sector register. Only the lower byte is used for DMA test. All outputs are asserted low.

Figure 2-20 23-143K5-00

T: PAL16R8
P: 23-143K5-00
M: HARDWARE ENGINEER
D: 10-JAN-86

S: PAL_CLK RX_D07 RX_D06 RX_D05 RX_D04 INTR_REQ /LD_CSR /RESET RX_D12
GND /TS_OE /INTR_DLY /INTR_DONE /DISK_WRITE /DISK_READ /LD_CSR_DLY
/DISK_RW /FORMAT PIN19 VCC

B: DISK_READ = LD_CSR • /RX_D12 • RX_D07 • /RX_D06 • /RX_D05 • /RESET
• LD_CSR • /RX_D12 • RX_D07 • RX_D06 • /RX_D05 • /RX_D04 • /RESET
• LD_CSR • /RX_D12 • RX_D07 • RX_D06 • RX_D05 • /RX_D04 • /RESET
• DISK_READ • LD_CSR_DLY

DISK_WRITE = LD_CSR • /RX_D12 • RX_D07 • /RX_D06 • RX_D05 • /RESET
• LD_CSR • /RX_D12 • RX_D07 • RX_D06 • RX_D05 • RX_D04 • /RESET
• DISK_WRITE • LD_CSR_DLY
• FORMAT • /INTR_DONE

DISK_RW = DISK_READ
• DISK_WRITE • /INTR_REQ • /INTR_DLY • /INTR_DONE

LD_CSR_DLY = LD_CSR • /LD_CSR_DLY • /RESET
• LD_CSR_DLY • /INTR_REQ • /RESET

FORMAT = LD_CSR • /RX_D12 • RX_D07 • RX_D06 • RX_D05 • RX_D04 • /RESET
• FORMAT • /INTR_DONE • /RESET

INTR_DLY = INTR_REQ
INTR_DONE = INTR_DLY • /INTR_REQ

E: E129, FUNCDEC2 (Function Decode) PAL for RX33 Controller

PAL generates signals which establish the mode of operation of the disk read and disk write PALs. A read or write mode remains in effect from the time the command is loaded into the CSR until an interrupt or reset occurs. If a disk read or write is in effect then DISK_RW is used by the RX_CTL PAL to force the data buffer to be selected. All outputs are asserted low.

FUNDEC2 adds output signals FORMAT, INTR_DLY, and INTR_DONE and changes equations for DISK_WRITE and DISK_RW slightly. New equations solve problems which sometimes occur during formatting commands which could cause floppy controller to write bad parity to a single memory location if index pulse occurred while floppy was in the middle of a memory read cycle. New equations now cause controller to wait for trailing edge of interrupt instead of using leading edge to terminate the format operation.

This PAL is used on the L0117 (M.std2) module of the HSC70.

The base part number for this PAL is 23-000K5-04.

Figure 2-21 23-144K5-00

T: PAL1028
P: 23-144K5-00
E: HARDWARE ENGINEER
D: 7-JAN-86

S: PAL_CLK BYTE_REQ PIN3 MASTER /DISK_READ PIN6 PIN7 /DMA_TEST PIN9
GND /TS_OE /DMA_TEST_FLAG /STOP_DMA_TEST /LB_LOADED /DMA_WR_REQ
/RX_MASTER_RD /LD_XBUF_HB /LD_XBUF_LB PIN10 VCC

B: $RX_MASTER_RD = DISK_READ \cdot BYTE_REQ \cdot /LD_XBUF_LB \cdot /LD_XBUF_HB$
 $\cdot /DMA_TEST \cdot /DMA_TEST_FLAG \cdot /DMA_WR_REQ \cdot /MASTER$
 $\cdot DISK_READ \cdot DMA_TEST \cdot /LD_XBUF_LB \cdot /LD_XBUF_HB$
 $\cdot /DMA_WR_REQ \cdot /MASTER \cdot /STOP_DMA_TEST$
 $\cdot DISK_READ \cdot RX_MASTER_RD \cdot /LD_XBUF_LB \cdot /LD_XBUF_HB$

$LD_XBUF_LB = DISK_READ \cdot RX_MASTER_RD \cdot /LB_LOADED \cdot /LD_XBUF_LB$
 $LD_XBUF_HB = DISK_READ \cdot RX_MASTER_RD \cdot LB_LOADED \cdot /LD_XBUF_HB$
 $DMA_WR_REQ = DISK_READ \cdot RX_MASTER_RD \cdot LD_XBUF_HB \cdot /MASTER$
 $\cdot /DMA_WR_REQ$
 $\cdot DISK_READ \cdot DMA_WR_REQ \cdot /MASTER$

$LB_LOADED = DISK_READ \cdot RX_MASTER_RD \cdot LD_XBUF_LB \cdot /LB_LOADED$
 $\cdot DISK_READ \cdot LB_LOADED \cdot /LD_XBUF_HB$

$DMA_TEST_FLAG = DMA_TEST \cdot DISK_READ \cdot /DMA_TEST_FLAG$
 $\cdot DMA_TEST_FLAG \cdot DISK_READ$

$STOP_DMA_TEST = DMA_TEST_FLAG \cdot DISK_READ \cdot /DMA_TEST$
 $\cdot STOP_DMA_TEST \cdot DISK_READ$
 $\cdot DMA_TEST_FLAG \cdot MASTER$

E: E133, DISKRD2 (Disk Read) Sequencer PAL for RX33 Controller

This PAL generates signals which steer the data path during reads from the floppy disk. All outputs are asserted low.

DMA_TEST input may be used to generate DMA writes to memory without a floppy drive cabled up. This aids in debug, board test, and diagnostic fault isolation.

DISKRD2 pattern changes equation for RX_MASTER_RD so that low byte does not get corrupted when reading floppy while system is heavily loaded and DMA latency is very slow. Floppy data register now cannot be read into XBUF until prior bus cycle has completed.

This PAL is used on the L0117 (M.std2) module of the HSC70.

The base part number for this PAL is 23-000K5-04.

For Internal Use Only
NBC50/70 PAL EQUATIONS

Figure 2-22 23-2495-00

T: PAL16L8
P: 23-249J6-00
D: 28-AUG-85

S: /DISK_READ /DISK_WRITE /REG_RD PIN4 PIN6 PIN8 /DAT_EN ADR_EN
PIN9 GND PIN11 PIN12 PIN13 PIN14 /END_XBUF /XMT PIN17 PIN18
PIN19 VCC

B: IF [VCC] END_XBUF = DISK_READ • DAT_EN
IF [VCC] XMT = DISK_READ • ADR_EN
• DISK_WRITE • ADR_EN
• DISK_READ • DAT_EN
• REG_RD

E: E135, MISC (Miscellaneous) PAL for RX33 Controller

This PAL generates signals which control the transmit/receive logic, enable the transmit buffer during disk reads, and generate DMA requests for both disk reads and writes. All outputs are asserted low.

Figure 2-23 23-069K5-00

T: PAL16R8
P: 23-069K5-00
D: 20-SEP-85

S: CLK DISK /ENA /CLR UPPER S1 I1 S0 IO GND
/ENAB IORS_FOLLOW_0 IORS_FOLLOW_1 VALID_0 VALID_1
I_DET_POL_0 I_DET_POL_1 IORS_DET_SFOL_0 IORS_DET_SFOL_1 VCC

B: /IORS_FOLLOW_0 = CLR * /IO * /S0
/IORS_FOLLOW_1 = CLR * /I1 * /S1
/VALID_0 = /DISK * IORS_DET_SFOL_0 * /S0 * IORS_FOLLOW_0
* IORS_DET_SFOL_0 * IO * /IORS_FOLLOW_0 * CLR * /VALID_0 * /IO
/VALID_1 = /DISK * IORS_DET_SFOL_1 * /S1 * IORS_FOLLOW_1
* IORS_DET_SFOL_1 * I1 * /IORS_FOLLOW_1 * CLR * /VALID_1 * /I1
/I_DET_POL_0 = /IO * DISK * /I_DET_POL_0 * UPPER * ENA
* IORS_DET_SFOL_0 * DISK * /DISK * /IO * CLR * IORS_FOLLOW_0
* DISK * /I_DET_POL_0
* DISK * I_DET_POL_0 * S0 * /IORS_DET_SFOL_0
/I_DET_POL_1 = /I1 * DISK * /I_DET_POL_1 * UPPER * ENA
* IORS_DET_SFOL_1 * DISK * /DISK * /I1 * CLR * IORS_FOLLOW_1
* DISK * /I_DET_POL_1
* DISK * I_DET_POL_1 * S1 * /IORS_DET_SFOL_1
/IORS_DET_SFOL_0 = IO * DISK * /IORS_DET_SFOL_0 * /IORS_DET_SFOL_0
* DISK * S0 * UPPER * IORS_DET_SFOL_0 * ENA * DISK * /IO
* /VALID_0 * DISK * CLR * /IORS_FOLLOW_0 * DISK * /IORS_DET_SFOL_0
* /DISK * /S0
/IORS_DET_SFOL_1 = I1 * DISK * /IORS_DET_SFOL_1 * /IORS_DET_SFOL_1
* DISK * S1 * UPPER * IORS_DET_SFOL_1 * ENA * DISK * /I1
* /VALID_1 * DISK * CLR * /IORS_FOLLOW_1 * DISK * /IORS_DET_SFOL_1
* /DISK * /S1

E: These equations latch the occurrence of sector and index pulses
so that the K edi microcode can deal with them via polling

(Continued on next page)

66

Figure 2-23 (Cont.) 23-049KS-00

This PAL produces four output variables for each of two drives. These variables are

IOBS_FOLLOW - This is the OR of the drive's Sector Pulse and Index Pulse and is used to do edge-detection on the pulses.

IOBS_DET_SFOL - This is the "I saw something" signal from the PAL to the K edi microcode. None of the other outputs of the PAL are examined unless this bit is set. It is set on the falling edge of either Sector or Index Pulse, and cleared whenever it is read by K edi (while set) or when the PAL is cleared or when the VALID bit goes away.

VALID - This indicates if the PAL outputs are valid - i.e., if VALID is not asserted then the PAL saw a second pulse before the K edi microcode read the results of the previous pulse. It is set on the rising edge of Index Pulse, and cleared whenever the PAL is cleared or whenever a falling edge of Sector Pulse or rising edge of Index Pulse is detected with IOBS_DET_SFOL already set.

I_DET_FOL - this bit latches the fact that we have seen the rising edge of Index Pulse and the K edi has not yet been informed. It is set on the rising edge of Index Pulse, and cleared if the PAL is cleared or if the PAL is read while IOBS_DET_SFOL is set. It is also cleared if Sector Pulse is seen and IOBS_DET_SFOL has not been set - this overcomes a bug caused by the periodic disabling of drive state receivers by periodic diagnostics in the K edi microcode.

When this PAL is being used by K sti instead of K edi, the I_DET_FOL bit is simply a latched copy of Index Pulse and the IOBS_DET_SFOL bit is simply a latched copy of sector pulse

CHAPTER 3 HSC50 BOOT PROM LISTINGS

This chapter contains the Boot PROM listings for the HSC50. The page numbers for this chapter are in the upper right hand corner of the listings.

For Internal Use Only

MCS50 P.10C ROM BOOT V01-00 MACRO M200 16-MAY-82 11:16

TABLE OF CONTENTS

2-	27	#### MCS50 P.10C ROM Boot version V01-00 ####
4-	142	Useful Information
5-	199	Useful Information (continued)
6-	231	Program Overview
7-	269	Definitions
7-	290	Address Definitions
8-	336	Bit Definitions
11-	461	Alternate Entry Points and Pointer Table
13-	549	..TEST 0 - F-11 ISP Test
23-	1097	MEMORY TESTS
24-	1140	..TEST 1 - Program Memory (Swap Bits)
25-	1174	..TEST 2 - Program Memory (Vector area)
26-	1224	..TEST 3 - Program Memory (Partition area)
27-	1279	FALST Lamp Routine
28-	1305	..TEST 4 - TUSB Tests
32-	1559	..TEST 5 - Transfer Control to Loaded Image
35-	1690	Issue TUSB Sync and Self-Test
36-	1805	Read TUSB Boot Blocks to Program Memory
38-	1869	TUSB Read Routine
39-	1900	Synchronize with a TUSB
41-	2059	Send Character to TUSB
42-	2117	Receive a Character from a TUSB
43-	2187	Send a TUSB Command Packet
46-	2330	Receive a Command or Data Packet from the TUSB
53-	2652	Routine to save TUSB error reasons
54-	2686	Moving Inversions Test Description
55-	2726	Moving Inversions Test
56-	2833	Permanent Storage

```

22      ;+
23      ;*****
24      ;program name: HDC30 ROM Bootstrap
25      ;
26      ;
27 000000      .ident 01,00,( ROM Boot)
                .title HDC30 P.ioc ROM Boot V01-00
                .abttl ***** HDC30 P.ioc ROM Boot version V01-00 *****
                .ident /V0100/

28      ;
29      ;program abstract:
30      ; This program is designed to execute from the bootstrap ROM located
31      ; on the P.ioc module. The bootstrap consists of tests of the F-11
32      ; CPU, part of the Program P.ioc memory, and the TUS0; followed by
33      ; loading of the next part of the bootstrap sequence from the TUS0.
34      ; The purpose of the bootstrap tests is to provide confidence that
35      ; the P.ioc can be used to load further software from the TUS0.
36      ; The software loaded as the last action of the bootstrap program
37      ; consists of more extensive tests of the P.ioc module. After
38      ; these tests finish execution, the HDC30 Functional Software is
39      ; loaded and initiated.
40      ;
41      ; Two version of the boot ROM can be created from this source:
42      ; 1) The P.ioc version
43      ; 2) The P.IXX (Secondary P) version (define SECNDP)
44      ;
45      ;copyright statement:
46      ; Copyright 1982, Digital Equipment Corporation
47      ;
48      ;disclaimer:
49      ; The information in this document is subject to change without
50      ; notice and should not be construed as a commitment by Digital
51      ; Equipment Corporation. Digital Equipment Corporation assumes
52      ; no responsibility for any errors that may appear in this doc-
53      ; ument.
54      ;
55      ; The software described in this document is furnished under a
56      ; license and may only be used or copied in accordance with the
57      ; terms of such license.
58      ;
59      ; Digital Equipment Corporation assumes no responsibility for the
60      ; use or reliability of its software on equipment that is not sup-
61      ; plied by Digital.
62      ;
63      ;author - version - date list
64      ;
65      ; Mike Mare      V01-00  SEP 1982
66      ;
67      ;
68      ;.call hminit,lg0
69 000000      hminit
70 000000      lg0

```

```

72      |version history
73      |
122     | V01-00      15-SEP-82
123     |          1) Use Kernel PAR set for bootstrap temporary storage.
124     |          2) # of blocks loaded by boot is now 8 vs. 12
125     |          3) New Fault codes
126     |          4) New ROM entries added: Receive Packet, Send Packet
127     |          5) Don't set ROM vector until memory test
128     |          6) Added missing branch to MFS test
129     |          7) Turn off PMU before using PAR's for storage
130     |          8) Modified Fault lamp reporter. Now don't monitor Fault switch
131     |             after it is pressed once.
132     |          9) Added conditional 'SECOP' to allow assembling a P.xxx
133     |             (Secondary P) version of the bootstrap ROM:
134     |             a) Set 'Select P' bits to 11(2).
135     |             b) Puts 'Secondary P' code in Status Register
136     |          10) Now do a RESET to clear any pending I interrupts.
137     |          11) Minor code compression.
138     |
139     |-----
140     |-
```

Still Useful Information

- 1) **MCBIEM PROGRAM SIZE** - This program must fit into a 2,048 byte ROM.
(Locations 17773000 thru 17776777)
- 2) **Special Assembly Files Required** - This program must be assembled with the file 'MCBIEM.MAC/AT' to supply I/O page addresses.
- 3) **Special use of PWR PWR's (page address registers)** - PWR PWR's are used on a convenient plane to store information that must be passed along to the Init P.loc test which succeeds the boot, and are used as test locations. The contents of the USER PWR set are preserved. This is where boot reasons and failing addresses are stored.

FOR RELEASE ONLY

USER	KERNEL
0 preserved	TEST 0
1 preserved	initial stack
2 preserved	initial stack
3 preserved	initial stack
4 preserved	initial stack/register bit tests
5 preserved	unit 0 & type of boot (R3 on entry)
6 preserved	CSR of TUCB to use (R4 on entry)
7 preserved	Validation of unit/type/CSR

4) Warm Boot entry restrictions: When using the 'WARM' boot feature, the ROM must be entered with interrupts disabled (PRI=7), with the IRI off, or in Kernel mode with the I/O page mapped.

5) Selection of load device - To force this program to select a particular TUSB controller and unit for booting purposes, enter the ROM with the following parameters:

- a) R3 (lower byte) = Unit 0 of TUCB to be used for host.
(NOTE: unit byte may only be '0' or '1')
(upper byte) = Reserved, must be zero

- b) R4 (16 bits) = COR address of TUS0 to use for boot
c) R5 (16 bits) = Validation of unit and COR (R3 XOR R4)

If the data in R3 and R4 is valid ($R3 \text{ XOR } R4 = R5$) then the boot will first be attempted from the specified device. If this load attempt fails, or if the data in R3 and R4 is not valid, then the default sequence of TUSB selection will be used. (see label 't50tst' for a definition of the default TUSB selection sequence.)

6) Processor-time-dependent loops - All instruction sequences labeled 'PTIME*n*' are timing loops that depend on the basic cycle time of the F-11 CPU chip.

7) ROM VERSION NUMBER - The Version and Edit numbers of the ROM software are stored in the last word of the ROM (address 176776). The low byte is the version #, the high byte is the edit #.

8) **SECONDARY P VERSION** - define 'SECOP' to produce a Secondary P best.

66

199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229

.sbt11 Useful Information (continued)

~~~~~

0) TUSB ERROR CODE TABLE - This program constructs an error table which remembers why a particular TUSB unit could not be used for booting. If the bootstrap fails, and the error code displayed in the fault lamp is 'TUSB Failure', the error code table can be examined with 'F-11 OUT' to determine why none of the TUSB's were suitable for booting. The error code table begins at address 000400 in program memory and is laid out as follows:

| Address | Controller RCR | unit 0 |
|---------|----------------|--------|
| 000400  | 177520         | 0      |
| 000402  | 177520         | 1      |
| 000404  | 177520         | 0      |
| 000406  | 177520         | 1      |
| 000410  | 177560         | 0      |
| 000412  | 177560         | 1      |

The low byte of each word in the table contains a code which indicates the reason why the bootstrap software rejected that particular TUSB unit. (The failure codes are defined in the 'Bit Definitions' section of this document). The upper byte of each word in the table contains the error reason from the last end packet received from the TUSB (if any). Note that if a TUSB controller fails to synchronize, or fails its self-test, the software failure code will be stored in unit 0's word of the error table entry, and unit 1 will not be tried.

~~~~~

```

231      .title Program Overview
232      ;+
233      ;=====
234      ; Test the hardware of the P.10c module, including the F-11,
235      ; Program Memory ( 9 Kwords), and a TUS0.
236      ;
237      ;inputs: 1) Unit 0 of TUS0 to be used for boot. (R3, lower byte)
238      ;         2) CSR of TUS0 to be used for boot (R4)
239      ;         3) Validation word for unit and CSR (R5)
240      ;         4) type of boot (enter at 173000 for cold boot,
241      ;            173006 for warm boot.
242      ;
243      ;process: This test is initiated each time the P.10c subsystem is booted
244      ;            or rebooted. Different entry points to the ROM allow the software
245      ;            to determine whether the subsystem was being booted or rebooted.
246      ;            Testing begins with the ISP of the F-11 processor. All basic
247      ;            PDP-11 instructions, and all eight addressing modes are tested.
248      ;            Following the F-11 ISP tests, 9 Kwords of the P.10c Program memory
249      ;            are tested for proper addressing and ability to hold data and
250      ;            proper parity. The final test locates a working TUS0 tape drive
251      ;            that contains a HSC50 System Cassette, and reads the first 8 blocks
252      ;            from the tape to the 8 Kword partition in the Program memory.
253      ;
254      ;outputs:(to 'Init P.10c Test')
255      ;
256      ; case 1 - F-11 test passes, 8 Kword partition ok, TUS0 ok
257      ;           R0 = unit 0 of TUS0 used for boot (lower byte)
258      ;           type of boot (upper bit of upper byte) 0=cold, 1=warm
259      ;           R1 = CSR address of TUS0 used for boot
260      ;           R2 = Base address of tested 8 Kword partition
261      ;           'INIT' lamp is on.
262      ;
263      ; case 2 - F-11 test fails
264      ;           No parameters passed. F-11 'hangs' immediately. (branch self)
265      ;           'Init' lamp does not light, 'fault' lamp off.
266      ;
267      ; case 3 - No working 8 Kword partition found in Program Memory
268      ;           (or first 2048 bytes of Program Memory are faulty)
269      ;           'Init' lamp is lighted
270      ;           'Fault' lamp is lighted
271      ;           Depressing fault switch causes blinking fault code display
272      ;           'Fault code' = (memory test failure)
273      ;           HSC50 must be rebooted to attempt recovery
274      ;           Program Memory module probably needs replacing
275      ;
276      ; case 4 - No working TUS0 found with bootable media mounted
277      ;           'Init' lamp is lighted
278      ;           'Fault' lamp is lighted
279      ;           Depressing fault switch causes blinking fault code display
280      ;           'Fault code' = (TUS0 test failure)
281      ;           HSC50 must be rebooted to attempt recovery
282      ;           No TUS0 has system cassette mounted, or TUS0's are faulty
283      ;           (TUS0 error code table contains reason why each TUS0 unit
284      ;           failed. See 'TUS0 Error Table' under 'USEFUL INFORMATION'(above)
285      ;
286      ;=====
287      ;-
  
```



```

289 .cbl1 Definitions
290 .cbl1 Address Definitions
291
292 ;+
293 ;*****
294 ;
295 ; Address definitions
296 ;
297 ;*****
298 ;-
299     173000    bccode=173000    ;first byte in ROM
300     176777    lccode=176777    ;last byte in ROM
301     172340    ORFADR= 172340    ;address of kernel PAR 0
302     172340    TESTID= ORFADR    ;kernel PAR 0 is used to hold the test number
303
304 ;+
305 ; TUSB ERROR TABLES
306 ;
307 ; There are two words for each possible TUSB controller in this
308 ; table. In each of the two-word groups, the first word is for
309 ; unit 0 of that controller, the second word for unit one.
310 ;
311 ; The low byte of each word in the table is for remembering the
312 ; software reason that a particular TUSB unit was rejected. The high
313 ; byte of each word in the table, will hold the TUSB error code from
314 ; a radial-serial-protocol end packet (if any). A zero in either
315 ; byte indicates no specific error was identified.
316 ;-
317
318 ;low byte = ROM software error code, upper byte = end packet failure code (if an
319     000400    SL1ER0= 400        ;TUSB 01, unit 0
320     000402    SL1ER1= SL1ER0+2    ;TUSB 01, unit 1
321     000404    SL2ER0= SL1ER1+2    ;TUSB 02, unit 0
322     000406    SL2ER1= SL2ER0+2    ;TUSB 02, unit 1
323     000410    SL0ER0= SL2ER1+2    ;console, unit 0
324     000412    SL0ER1= SL0ER0+2    ;console, unit 1
325
326 ;miscellaneous temporary R/W locations
327     000414    CKSPX= SL0ER1+2    ;suspected radial serial packet checksum
328     000416    CKSPAC= CKSPX+2    ;actual radial serial packet checksum from TUSB
329     000420    TUERR0= CKSPAC+2    ;high byte = end packet code, low = software error code
330     000420    TUBER= TUERR0      ;software error code byte
331     000421    TUBER= TUERR0+1    ;hardware error code from end packet
332     000422    TUERTP= TUBER+2    ;ptr to current word of TUSB error table
333 000000
334     173000

```

```

336      .sect1 Bit Definitions
337
338      ;+
339      ;-----
340      ;
341      ;      Bit definitions
342      ;
343      ;-----
344      ;-
345
346      ;+
347      ;-----
348      ;      Miscellaneous definitions
349      ;
350      ;-----
351      ;-
352
353      TSMTHD= 72.      ;TUSB read response timer value( 72 times .55 sec = 40 sec)
354      PMSIZ= 16384.    ;size of program memory partition to be tested in bytes.
355      VECsiz= 2048.    ;size of vector area to be tested in bytes.
356      LODSIZ= 8.      ;# of blocks to load for boot
357
358      ;+
359      ;-----
360      ;
361      ;      Fault register code definitions
362      ;
363      ;-----
364      ;-
365
366      piofal=21        ;P.10C module failure
367      memfal=22        ;memory module failure
368      tusbfl=23        ;TUSB failure code
369
370      ;+
371      ;-----
372      ;
373      ;      TUSB code definitions
374      ;
375      ;-----
376      ;-
377
378      ;Flag Byte Codes
379      init=4           ;TUSB Initialize character
380      cmd=2            ;TUSB Command identifier
381      data=1          ;TUSB Data identifier
382      cont=20          ;TUSB Continue character
383
384      ;Op-codes
385      read=2           ;read command
386      write=3          ;write command
387      posite=3         ;position tape command
388      diag=7           ;diagnose command
389      endp=100         ;end packet op code
390      STUBRE=1         ;NOTE: no longer defined in MSCDEF (14-aug-80)
  
```

```

391
392
393
394
395
396
397
398
399
400      100000      TERPD=100000      ;1 when terminal enable, 0 when Secure
401      004000      SAMP0=004000      ;1 when Swap memory boards
402      002000      SAMPB=002000      ;1 when Swap memory banks
403      001000      SELP1=001000      ;Bit 1 of P.10c interrupt code
404      000400      SELP0=000400      ;Bit 0 of P.10c interrupt code
405      000200      ENARB=000200      ;1 when enable cmm arbitration
406      000100      CLABIT=000100      ;1 when clear I/O page device registers
407      000040      HIFTEST=000040      ;1 when enable high byte parity test
408      000020      LOPTST=000020      ;1 when enable low byte parity test
409      000010      LEDBIT=000010      ;1 when light STATE LED
410      000004      IOWBIT=000004      ;1 when create a non-memory-access cycle
411      000002      CNTIE=000002      ;1 when control memory interrupts enabled
412      000001      LCKBIT=000001      ;1 when create a control memory lock cycle
413
414
415
416
417
418      TUSB Software Error codes
419      (used in low byte of TUSB error table words)
420
421
422      000001      TUBOP= 1      ; 1 = 1001 trap while accessing TUSB CSR's
423      000002      TUSYN= 2      ; 2 = TUSB synchronization sequence failed
424      000003      TUBLFT= 3      ; 3 = TUSB self-test failed
425      000004      TUSOFT= 4      ; 4 = TUSB unit did not contain bootable image
426      000005      TUCKSM= 5      ; 5 = Checksum error in (received) radial-serial-packet
427      000006      TUBTTO= 6      ; 6 = timeout on XMIT RDY when writing to TUSB
428      000007      TURCTO= 7      ; 7 = timeout on RCRD DONE when receiving from TUSB
429      000010      TURCER= 10      ;10 = UNIT error on received character (overrun, framing error)
430      000011      TUPROT= 11      ;11 = radial-serial-packet protocol error (unknown packet type)
431
432
433
434      Special local definition of POP-11 RESET instruction
435      (Required since RESET is illegal in normal NDC50 code,
436      and NDC50 Macro Library defines RESET to cause an error)
437
438
439
440      000005      RESETS= 000005      ;define our own mnemonic for a RESET instruction
  
```

442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459

004000
 002000
 001000
 000400
 000020
 000010
 000004
 000002
 000001
 000037

P.IOC Switch/Display Register Bit Definitions

FLTM =004000 ;1 when 'Fault' switch depressed
 BLKM =002000 ;1 when 'Online' switch depressed
 HSM =001000 ;1 when 'First Unmarked' switch depressed
 HSM =000400 ;1 when 'Second Unmarked' switch depressed
 INIUP=000020 ;1 when light 'Init' lamp
 FLTUP=000010 ;1 when light 'Fault' lamp
 BLKUP=000004 ;1 when light 'Online' lamp
 HBUUP=000002 ;1 when light 'First Unmarked' lamp
 HBUUP=000001 ;1 when light 'Second Unmarked' lamp
 ALLUP=INIUP!HBUUP!BLKUP!FLTUP!INIUP ;inclusive OR of all lamp bits

```

461                                     .sbt11 Alternate Entry Points and Pointer Table
462
463                                     {+
464                                     ;
465                                     ;
466                                     ; Alternate entry points to this program are supplied to allow
467                                     ; the distinction between a 'cold' boot and a 'warm' boot.
468                                     ; (A cold boot is the result of power-up, or '173000G' to ODT. No
469                                     ; saved context exists for a cold boot. A warm boot indicates that
470                                     ; there is saved context in the User PAR set, to be used by the Init
471                                     ; P.10C test.)
472                                     ;
473                                     ;inputs:
474                                     ; R3 = unit 0 of TUSB to use for boot (lower byte)
475                                     ; R4 = CSR address of TUSB to use for boot (16 bits)
476                                     ; R5 = Validation of unit/CSR (R3 XOR R4)
477                                     ; For Warm boots, the User PAR set contains the reason
478                                     ; for the reboot, and other context. This context is not
479                                     ; used by the Bootstrap ROM, it is merely not disturbed.
480                                     ;
481                                     ;process:
482                                     ; Set up a 'type of boot' parameter, depending upon which
483                                     ; entry into the ROM is used. The following entry points
484                                     ; are provided:
485                                     ; 1) Entry for 'Cold' boot
486                                     ; 2) Entry for 'Warm' boot
487                                     ;
488                                     ;outputs:
489                                     ; R3, R4, and R5 preserved
490                                     ; 'Type of boot' encoded in upper bit of R3
491                                     ;
492                                     ; The value of the upper bit of R3 is as follows:
493                                     ; cold boot = 0
494                                     ; warm boot = 1
495                                     ;
496                                     ;notes:
497                                     ; History of changes to this routine
498                                     ;
499                                     ;-----
500                                     ;-
501

```

```

503 173000      coldbt: ;+
504              |-----|
505              |
506              |      Entry for 'Cold' boot
507              |
508              |-----|
509              |
510              |
511 173000 106427 000340      mps  0340      ;prevents real time clock from interrupting
512 173004 000137 173032      jmp   FLITST      ;cold boot entry
513
514
515              ;+
516              |-----|
517              |
518              |      Entry for 'Warm' boot   (base address + 10)
519              |
520              |-----|
521              |
522              |
523 173010 052703 100000      BTYPE2: b1s  0100000,R3      ;set R3 negative to indicate warm boot
524 173014 000410              br    ALTENT      ;start bootstrap tests
525
526
527              ;+
528              |-----|
529              |
530              |      Table of pointers to ROM routines that are used from ROM
531              |      (by the INIT P.IOC test)
532              |
533              |      The relative position of these pointers, with respect to the
534              |      starting address of the ROM, must NEVER be changed.
535              |
536              |-----|
537              |
538              |
539 173016 175630      TUSBY:  .word  TSYSC      ;base + 16      TUSB synchronization routine
540 173020 175442      TUSRD:  .word  TSDRD      ;base + 20      TUSB read routine
541 173022 176136      TUSBN:  .word  SENDPK     ;base + 22      Send TUSB command packet
542 173024 176370      NINENT: .word  MOVINU     ;base + 24      Moving Inversions ROM test
543 173026 174372      ERRAPT: .word  ANYFAL     ;base + 26      Fault code display routine
544 173030 176242      TUSBC:  .word  RCDUPK     ;base + 30      Receive TUSB command or data packet
545
546              ;***** END OF POINTER TABLE *****
547

```


349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404

```

..tbl1 ..TEST 0 - F-11 ISP Test
;+
;~~~~~
; Test the basic FBP-11 instruction set and addressing modes.
;
;Inputs:      FPM Priority field = 7 (interrupts disabled) upon entry
;              R3 = Unit 0 to use for boot (lower byte, value of 0 or 1 only)
;              R4 = CSR address of TUS0 to use for boot (16 bit I/O page address)
;              R5 = Validation for R3 and R4 (R3 XOR R4)
;                  (validation prevents spurious device selection)
;
;Process: Test the Instruction Set Processor (ISP) of the P.10c's F-11
;          processor.
;          Instructions tested:
;              CLR(B),CMR(B),INC(B),DEC(B),NEG(B),TST(B),ROR(B),ROL(B),
;              AND(B),ASL(B),SHAR,NOP,ADC(B),CCC,CLM,CLV,CLZ,SCC,SEN,SEV,SEZ
;
;              MVA(B),DVP(B),BIT(B),BIC(B),BIS(B),ADD,SUB
;
;              RR,BAC,REG,BPL,BMI,BCC(BHIS),BCS(BLD),BNE
;              BLT,BGT,BLE,BHI,BLOS,BVC,BVS
;
;              JRP,JRR,RTS,SEN,HTPS,HFPA
;
;          Addressing modes tested:
;              All eight addressing modes are tested
;
;          Instructions NOT tested:
;              ARH,ARHC,BPT,DIV,DNT,MULT
;              IOT,JMRR,HFPI,HTPI,MUL,RESET,RTI,RTT,SBC
;              SPL,TRAP,MATT,XOR
;
;          F-11 Logic NOT tested:
;              Memory Mapping, Interrupt priority and arbitration,
;              Stack Overflow, Power Fail, Bus Time-out Logic
;
;          P.10c logic NOT tested
;              Control Memory Windows, ECC tables
;              Operator console interface, 'K' status registers
;              Control and arbitration of Data and Control buses
;              Remainder of Program Memory (total size minus 8 Kwords)
;
;Outputs:
;          Case 1 - no failure detected
;              'Init' lamp is lighted
;              Control passed to Program Memory Test
;              R3,R4, and R5 are preserved in MPU Kernel P0R's 5,6, and 7
;
;          Case 2 - failure detected
;              F-11 CPU is 'hung'. PC of 'hung' identifies failure.
;              'Init' lamp is out, 'Fault' lamp is out
;
;Notes:
;History of changes to this test:
;~~~~~
;-

```

```

606 173032
607
608
609
610
611
612
613
614 173032 042703 100000
615 173036 000005
616 173040 005037 172040
617 173044 005006
618 173046 005106
636
637
638
639
640
641
642
643
644
645
646
647
648 173050 000277
649 173052 001377
650 173054 100377
651 173056 103377
652 173060 003377
653 173062 101377
654 173064 002777
655 173066 102377
656 173070 000250
657 173072 002377
658 173074 000270
659 173076 002777
660 173100 000241
661 173102 103777
662 173104 000261
663 173106 103377
664 173110 000244
665 173112 001777
666 173114 000264
667 173116 001377
668 173120 000242
669 173122 102777
670 173124 000262
671 173126 102377
672 173130 000257
673 173132 001777
674 173134 100777
675 173136 103777
676 173140 002777
677 173142 003777
678 173144 101777
679 173146 102777
  
```

```

F11TST: j+
-----
;
; Initialize stack pointer such that if a trap or interrupt
; occurs, the F-11 will generate a double bus error
;
-----
;
;
ALTERN: RESETS      bic      0100000,R3      ;set 'type of boot' to 'cold boot'
;                                ;clear all OCP lamps, clear all I interrupts
;                                ;set test 0 to 0
;                                ;SP gets a 0
;                                ;SP gets 177777, causes DBL BUS ERR if trap occurs
;
-----
;
; Test Branch and Condition Code Instructions
;
; Test BR, BNE, BEQ, BPL, BMI, BCC(BNIS), BCS(BLO)
;     BGE, BLT, BGT, BLE, BHI, BLOS, CCC, SCC
;     SEN, SWS, SVC, SDN, CLM, SEZ, CLZ, SEC, CLC
;
-----
;
;
; M (- 1, Z (- 1, V (- 1, C (- 1
; nbe (no branch expected)
; nbe
; nbe
; nbe
; nbe
; nbe
; nbe
; nbe
; n (- 0
; nbe (n clear, v set)
; n (- 1
; nbe (n set, v set)
; c (- 0
; nbe
; c (- 1
; nbe
; z (- 0
; nbe
; z (- 1
; nbe
; v (- 0
; nbe
; v (- 1
; nbe
; M (- 0, Z (- 0, V (- 0, C (- 0
; nbe (no branch expected)
; nbe
; nbe
; nbe
; nbe
; nbe
; nbe
  
```

600 173150 000401	br	10	;branch forward
601 173152 000777	br	.	;failed to branch forward
602 173154 000270	10: set		;set the N bit (N (- 1)
603 173156 100377	hpi	.	;hpi (no branch expected)
604 173158 002377	hpi	.	;hpi
605 173162 002377	hpi	.	;hpi
606 173164 002402	bit	20	;should branch
607 173166 000777	br	.	;previous branch failed
608 173170 003402	30: bti	100	;should branch
609 173172 100776	20: bti	30	;should branch backwards
610 173174 000777	br	.	;backwards branch failed
611			
612			
613 173176	100: jt		
614			
615			
616			
617			
618			
619			
700			
701			
702			
703			
704 173176 005000	clr	R0	; R0 (- 0, C (- 0
705 173200 005100	com	R0	; R0 (- 177777, C (- 1
706 173202 005200	inc	R0	; R0 (- 000000, C = 1 (C not affected by inc)
707 173204 005300	dec	R0	; R0 (- 177777, C = 1 (C not affected by dec)
708 173206 000240	nop		;ensure nop does not interfere
709 173210 100377	bcc	.	;hang here if c bit is clear, or bcc fails
710 173212 005400	neg	R0	; R0 (- 000001, C (- 1
711 173214 100377	bcc	.	;hang here if c bit is clear, or bcc fails
712 173216 005700	tst	R0	; R0 = 000001, C (- 0
713 173220 001777	beq	.	;hang here if R0 = 0, or tst or beq fails
714 173222 006000	ror	R0	; R0 (- 000000, C (- 1
715 173224 001377	bne	.	;hang here if R0 not 0
716 173226 100377	bcc	.	;hang here if C didn't get loaded with a 1
717 173230 006100	rol	R0	; R0 (- 000001, C (- 0
718 173232 000240	nop		;ensure nop does not interfere
719 173234 100377	bcs	.	;hang if c bit is set
720 173236 001777	beq	.	;hang if R0 is 0
721 173240 000261	sec		; R0 = 000001, c (- 1
722 173242 000300	and	R0	; R0 (- 000400, C (- 0
723 173244 100377	bcs	.	;hang if and did not clear c bit
724 173246 105700	tsb	R0	;test for lower byte of R0 = 0
725 173250 001377	bne	.	;hang here if tsb failed, or R0 (lower) ne 0
726 173252 000300	and	R0	; R0 (- 000001, C (- 0
727 173254 105700	tsb	R0	;test that lower byte of R0 now not zero
728 173256 001777	beq	.	;hang if tsb failed, or R0 (lower) = 0

— — — — —

Address mode 0, using R0, R1, and SP (R6)

```

mov    SP, R0      ; R0 (- 177777
cmp     SP, R0      ; R0 = 177777, SP = 177777
bne     .           ;hang if SP and R0 not equal
bic     SP, R0      ; R0 (- 000000
bne     .           ;hang if R0 not 0
cmp     SP, R0      ; R0 = 000000, SP = 177777
bneq    .           ;hang if compare failed
inc     R0          ; R0 (- 000001
add     SP, R0      ; R0 (- 000000, C (- 1
bne     .           ;hang if add failed
bnc     .           ;hang if c bit did not set
bis     SP, R0      ; R0 (- 177777
cmn     R0          ; R0 (- 000000
bne     .           ;hang if R0 not zero
sub     SP, R0      ; R0 (- 000001
bneq    .           ;hang if R0 = 0
dec     R0          ; R0 (- 000000
bne     .           ;hang if R0 not 0
inc     R0          ; R0 (- 000001
clr     R1          ; R1 (- 000000
inc     R1          ; R1 (- 000001
bic     R0, R1      ; R1 (- 000000
bne     .           ;hang if R1 not 0
inc     R1          ; R1 (- 000001
cmn     R0          ; R0 (- 177776
add     R1, R0      ; R0 (- 177777, C (- 0
bcs     .           ;hang if c bit is set
cmn     R0          ; R0 (- 000000
bne     .           ;hang if R0 not 0

```

771			
772			
773			
774			
775			
776			
777			
778			
779			
780			
781			
782 173352	005100	com	R0
783 173354	110001	movb	R0,R1
784 173356	005101	com	R1
785 173360	001377	lne	.
786 173362	110100	movb	R1,R0
787 173364	001377	lne	.
788 173366	105100	comb	R0
789 173370	105700	tstb	R0
790 173372	001777	beq	.
791 173374	000300	movb	R0
792 173376	105700	tstb	R0
793 173400	001377	lne	.
794 173402	005101	com	R1
795 173404	103001	clrb	R1
796 173406	105701	tstb	R1
797 173410	001377	lne	.
798 173412	000301	movb	R1
799 173414	105701	tstb	R1
800 173416	001777	beq	.
801 173420	120001	cmph	R0,R1
802 173422	001777	beq	.
803 173424	000300	movb	R0
804 173426	106300	aslb	R0
805 173430	103377	bcc	.
806 173432	120001	cmph	R0,R1
807 173434	001777	beq	.
808 173436	105200	incb	R0
809 173440	006300	asl	R0
810 173442	105200	incb	R0
811 173444	120001	cmph	R0,R1
812 173446	001377	lne	.
813 173450	105300	decb	R0
814 173452	106200	asrb	R0
815 173454	120001	cmph	R0,R1
816 173456	001377	lne	.
817 173460	140100	bicb	R1,R0
818 173462	001377	lne	.
819 173464	150100	bicb	R1,R0
820 173466	105100	comb	R0
821 173470	001377	lne	.
822 173472	130100	bitb	R1,R0
823 173474	001377	lne	.
824 173476	105200	incb	R0
825 173500	105101	comb	R1
826 173502	105201	incb	R1
827 173504	130100	bitb	R1,R0

```

;-----
;
; Test CLRD, COMB, INCB, DECB, NEGB, TSTB, RORB, ROLB
; RORL, RLLB, RLAB, ORB, ORL, ORB, ORB, ORB, ORB
;
; Address mode 0. Using R0 and R1
;-----
;
; R0 (- 177777
; R1 (- 177777 (movb to register sign extends)
; R1 (- 000000
; hang if R1 not 0
; R0 (- 000000 (movb to register sign extends)
; hang if R0 not 0
; R0 (- 000377
; R0 = 000377
; hang if lower byte is zero
; R0 (- 177400
; R0 = 177400 (lower byte = 0)
; hang if lower byte of R0 not 0
; R1 (- 177777
; R1 (- 177400
; R1 = 177400
; hang if lower byte of R1 not 0
; R1 (- 000377
; R1 = 000377
; hang if lower byte of R1 = 0
; R0 = 177400, R1 = 000377
; hang if lower bytes are equal
; R0 (- 000377
; R0 (- 000376, C (- 1
; hang if c bit is not set
; R0 (lower) = 376, R1 (lower) = 377
; hang if lower bytes are equal
; R0 (- 000377
; R0 (- 000776
; R0 (- 000777
; R0 = 000777, R1 = 000377 (equal in lower byte)
; hang if lower bytes are not equal
; R0 (- 000776
; R0 (- 000777 (lower byte sign extends)
; R0 = 000777, R1 = 000377 (equal in lower byte)
; hang if lower bytes are not equal
; R0 (- 000400 (lower byte cleared)
; hang if lower byte did not result in a 0
; R0 (- 000777
; R0 (- 000400 (lower byte 0)
; hang if lower byte not 0
; R0 = 000400, R1 = 000377
; hang if bit test result was a 1
; R0 (- 000401
; R1 (- 000000
; R1 (- 000001
; R0 = 000401, R1 = 000001

```


000 173506 001777	beq	.	hang if bit test result was zero
009 173510 000241	cle	.	; C (- 0
030 173512 106001	rorb	R1	; R1 (- 000000, C (- 1
031 173514 103377	bcc	.	hang if c bit is not a 1
032 173516 001377	bne	.	hang if result in R1 not 0
033 173520 106101	rolb	R1	; R1 (- 000001, C (- 0
034 173522 103777	bcs	.	hang if c bit is set
035 173524 001777	beq	.	hang if result in R1 is 0
036 173526 105401	negb	R1	; R1 (- 000377, c (- 1
037 173530 105301	incb	R1	; R1 (- 000000, C remains 1 (not affected)
038 173532 103377	bcc	.	hang if c bit clear
039 173534 001377	bne	.	hang if result in R1 is not zero
040 173536 000300	subb	R0	; R0 (- 000401
041 173540 105300	decb	R0	; R0 (- 000400
042 173542 001377	bne	.	hang if result in R0 (lower) is not 0
043 173544 005700	tst	R0	; R0 = 000400
044 173546 001777	beq	.	hang if result in R0 is 0
045			
046			
047			
048			
049			
050			
051			
052			
053			
054			
055			
056			
057 173550 005000	clr	R0	; R0 (- 000000
058 173552 005001	clr	R1	; R1 (- 000000
059 173554 005002	clr	R2	; R2 (- 000000
060 173556 005200	inc	R0	; R0 (- 000001
061 173560 005201	inc	R1	; R1 (- 000001
062 173562 006300	asl	R0	; R0 (- 000002
063 173564 006301	asl	R1	; R1 (- 000002
064 173566 105301	subb:	decb R1	; R1 (- R1 minus 1
065 173570 105202	incb	R2	; R2 (- R2 plus 1
066 173572 077003	sub	R0,subb	; R0 (- R0 minus 1, branch if result not 0
067 173574 105701	tstb	R1	; R1 should now be zero also
068 173576 001377	bne	.	hang if R1 is not 0
069 173600 105302	decb	R2	; R2 (- 1
070 173602 105302	decb	R2	; R2 (- 0
071 173604 001377	bne	.	hang if R2 did not equal 2 after sub loop above
072 173606 000137 173614	jmp	jmpfar	test a forward jump
073 173612 000777	br	.	hang if jump failed
074 173614 000137 173630	jmpfar:	jmp jmptr1	test another forward jump
075 173620 000777	br	.	hang if jump failed
076 173622 000137 173636	jmptr2:	jmp F11000	test another forward jump
077 173626 000777	br	.	hang if jump failed
078 173630 000137 173622	jmptr1:	jmp jmptr2	test a backward jump
079 173634 000777	br	.	hang if jump failed


```

001 173636
002
003
004
005
006
007
008
009
010
011
012
013 173636 012700 173724
014 173642 011001
015 173644 022001
016 173646 001377
017 173650 020027 173726
018 173654 001377
019 173656 063001
020 173660 020027 173730
021 173664 001377
022 173666 163001
023 173670 044001
024 173672 001377
025 173674 020027 173724
026 173700 001377
027 173702 056001 000004
028 173706 037001 000006
029 173712 001777
030 173714 005101
031 173716 001377
032 173720 000403
033 173722 000777
034
035 173724 173724
036 173726 173724
037 173728 177777
038 173732 173730

```

```

F11A00: ;+
-----
;
;   Test addressing modes
;
;   Address modes 1, 2, 3, 4, 5, 6, and 7 are tested
;
;   R0 and R1 are used
;
-----
;-

```

```

mov    @moddt,R0      ;R0 gets pointer to data for this test (912)
mov    (R0),R1        ;R1 gets test data (911)
cmp    (R0)+,R1       ;check data against itself (912)
bne    .              ;hang if datum not equal
cmp    R0,@MODDAT+2    ;check R0 for correct contents
bne    .              ;hang if R0 is wrong
add    @-(R0)+,R1      ;add test data to R1 (913)
cmp    R0,@MODDAT+4    ;check R0 for correct value
bne    .              ;hang if wrong value in R0
sub    @-(R0),R1       ;subtract test data from R1 (915)
bic    -(R0),R1        ;clear R1 by bit clear with same data (914)
bne    .              ;hang if R1 was not cleared
cmp    R0,@MODDAT      ;check R0 for correct value
bne    .              ;hang if R0 is incorrect
bic    4(R0),R1        ; R1 (- 177777 (916)
bic    06(R0),R1       ;test for any bits set (917)
beq    .              ;hang if no bits are set
com    R1              ; R1 (- 000000
bne    .              ;hang if R1 was not all ones before com
br     F11ATP         ;transfer to next test
br     .              ;previous branch failed

```

```

moddt: .word moddt      ;data locations for addressing mode test
        .word moddt
moddt1: .word 177777
        .word moddt1

```

For Internal Use Only

WBC50 P.10C R01 0007 V01-00 P0C00 P0200 16-M0A-02 11:16 PAGE 19

..TEST 0 - F-11 ISP TEST

920 173734

921

922

923

924

925

926

927

928

929

930

931

932

933

934

935

936

937 173734 000277

938 173736 106700

939 173740 020027 177757

940 173744 001377

941 173746 000257

942 173750 106700

943 173752 020027 177740

944 173756 001377

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959 173760 003037 0000000

960 173764 012706 0000000

961

962 173770 004737 174002

963 173774 000777

964 173776 000405

965 174000 000777

966

967 174002 062716 000002

968 174006 000207

969 174010 000777

F11MTP: j+

Test MFS Instruction

Address mode 0

Note: Until interrupt logic is tested, (in 'Init P.10c test' following the R01 test), testing of the priority bits in the P01 is delayed to avoid uncontrolled detection of failures in the interrupt logic. This test restricts itself to checking that the priority is now 7, and that the N, Z, V, and C bits can be set and cleared.

```

; N (- 1, Z (- 1, V (- 1, C (- 1
; R0 (- 177757 (lower byte of P01, sign extend)
; check for pri 7, all cc bits set
;hang if mfs failed
; N (- 0, Z (- 0, V (- 0, C (- 0
; R0 (- 177740
; check for pri 7, all cc bits clear
;hang if mfs failed

```

Test JSR and RTS instructions

TURN OFF THE P01

NOTE: In order to test JSR, the stack must be used. This requires using some location in memory for a one word stack.

This test uses part of the P01's Kernel Page Address Register Set as a stack area. (8 contiguous 16 bit registers)

```

;TURN OFF THE P01
;point SP to Kernel P01S
;(KPAR's 4,3,2,1, and 0 used for stack)
;test JSR instruction. Return at call46
;hang if JSR returns to wrong location
;Correct return from JSR above
;hang if JSR returned to wrong location

JRTST: add    02,(SP)    ;update return address by 2
           rts    pc    ;test the RTS instruction
           br     .      ;hang if RTS failed to return

```

```

971 174012
972
973
974
975
976
977
978
979
980
981
982
983 174012 012701 0000000
984 174016 010321
985 174020 010421
986 174022 010511
987 174024 005001
988 174026 005301
989 174030 001113
990 174032 103512
991 174034 103501
992 174036 001110
993 174040 005101
994 174042 011101
995 174044 001105
996 174046 103104
997 174050 012702 177777
998 174054 103502
999 174056 103100
1000 174060 020227 177000
1001 174064 001075
1002 174066 000261
1003 174070 005501
1004 174072 103472
1005 174074 005301
1006 174076 001070
  
```

F11A0C: ;

```

-----
;
;      Test ADC and ADCB instructions
;
;      Move the boot parameters from R3, R4, and R5 to the Memory
;      Management Kernel PAR's 5, 6, and 7, where they will be
;      preserved across the memory and TUSB tests to follow.
;
-----
;
  
```

```

MOV    00UPAR5,R1      ;point R1 to Kernel par 5
MOV    R2,(R1)+         ;save unit and type of boot in UPAR5
MOV    R4,(R1)+         ;save CSR of TUSB in UPAR6
MOV    R5,(R1)          ;save validation word for unit and CSR
CLR    R1               ; R1 = 0, C = 0
ADC     R1               ; R1 = 0      (0+0=0)
BNE     CPUFAL           ;br if answer is non-zero
BCS     CPUFAL           ;br if c bit is set
ADCB   R1               ; R1 = 0      (0+0=0)
BNE     CPUFAL           ;br if answer is non-zero
CLR    R1               ; R1 = 177777, C = 1
ADC     R1               ; R1 = 0, C = 1 (177777+1=0)
BNE     CPUFAL           ;br if answer is not zero
BCS     CPUFAL           ;br if c bit did not set
MOV    0177777,R2       ; R2 = 177777, C = 1
ADCB   R2               ; R2 = 177000, C = 1
BCS     CPUFAL           ;br if c bit did not set
CMP     R2,0177000       ;check for correct answer in R2
BNE     CPUFAL           ;br if wrong answer
SETC    C               ; C = 1
MOV     R1              ; R1 = 1, C = 0 (0+1=1)
BCS     CPUFAL           ;br if c bit is set
CLR     R1              ; R1 = 0      (1-1=0)
BNE     CPUFAL           ;br if R1 did not contain a 1
  
```

For Internal Use Only

WBCSO P.10C ROM BOOT V01-00 MACRO M0200 16-MAY-82 11:16 PAGE 21

..TEST 0 - F-11 IOP TEST

```

1008
1009
1010
1011
1012
1013
1014
1015
1016 174100 012701 100000      mov    0100000,R1      ;preset all registers to 100000
1017 174104 010102      mov    R1,R2
1018 174106 010203      mov    R2,R3
1019 174110 010304      mov    R3,R4
1020 174112 010405      mov    R4,R5
1021 174114 010506      mov    R5,SP
1022 174116 001460      beq    CPUFAL          ;abort if bit was dropped along the way
1023 174120 010637 0000000    mov    SP,0KPAR4      ;set kernel PAR4 to same pattern
1024 174124 012700 000017      mov    015,R0          ;R0 will count 15 shifts
1025 174126 006201      100:  asr    R1          ;shift each register 1 bit right, sign extending 1's
1026 174132 006202      asr    R2
1027 174134 006203      asr    R3
1028 174136 006204      asr    R4
1029 174140 006205      asr    R5
1030 174142 006206      asr    SP
1031 174144 006237 0000000    asr    0KPAR4
1032 174150 020102      cmp    R1,R2          ;check for new pattern, with one additional '1' bit
1033 174152 001042      bne    CPUFAL
1034 174154 020304      cmp    R3,R4
1035 174156 001040      bne    CPUFAL
1036 174160 020506      cmp    R5,SP
1037 174162 001036      bne    CPUFAL
1038 174164 020637 0000000    cmp    SP,0KPAR4
1039 174170 001033      bne    CPUFAL
1040 174172 077022      sub    R0,100          ;branch until 15 shifts have been performed
1041 174174 023727 0000000 177777  cmp    0KPAR4,0177777  ;is final pattern all 1's ??
1042 174202 001026      bne    CPUFAL

```

```

1044                                     ;+
1045                                     |-----|
1046                                     |
1047                                     |   Basic Addr Test   |
1048                                     |
1049                                     |-----|
1050                                     |-
1051
1052 174204 012706 0000000 mov    00KPAR5,SP    ;set stack pointer to MPU Kernel PAR5
1053 174210 005001        clr    R1                ; R1 = 0
1054 174212 062701 125252 add    0125252,R1        ; R1 = 125252, N=1, Z=0, V=0, C=0
1055 174216 062701 052525 add    0052525,R1        ; R1 = 177777, N=1, Z=0, V=0, C=0
1056 174222 103016        bcs    CPUFAL            ;br if c bit is set
1057 174224 003015        bgt    CPUFAL            ;br if n bit is not set
1058 174226 102414        bvs    CPUFAL            ;br if v bit is set
1059 174230 062701 146314 add    0146314,R1        ; R1 = 146313, N=1, Z=0, V=0, C=1
1060 174234 103011        bcc    CPUFAL            ;br if c bit is not set
1061 174236 062701 125252 add    0125252,R1        ; R1 = 073565, N=0, Z=0, V=1, C=1
1062 174242 102006        bvc    CPUFAL            ;br if v bit did not set
1063 174244 103005        bcc    CPUFAL            ;br if c bit did not set
1064 174246 062701 104213 add    0-73565,R1        ; R1 = 000000, N=0, Z=1, V=0, C=1
1065 174252 001002        bne    CPUFAL            ;br if R1 not zero
1066 174254 102401        bvs    CPUFAL            ;br if v bit is set
1067 174256 103003        bcs    F1100N           ;br if c bit is set, all F-11 tests have passed
1068
1069
1070 174260 CPUFAL: ;+
1071                                     |-----|
1072                                     |
1073                                     |   F-11 CPU test failure. (only used for last few F-11 tests)
1074                                     |
1075                                     |   1) Set to RD to fault code for 'P.10c failure'
1076                                     |
1077                                     |   2) light the FAULT lamp
1078                                     |
1079                                     |   3) Monitor the FAULT switch
1080                                     |
1081                                     |-----|
1082                                     |-
1083
1084 174260 012700 000021 mov    0P10FAL,R0    ;R0 gets error code for P.10c failure
1085 174264 000542        br     ANYFAL            ;go to common error display routine
1086
1087 174266 F1100N: ;+
1088                                     |-----|
1089                                     |
1090                                     |   Light the 'Init' lamp to show F-11 test passed
1091                                     |
1092                                     |-----|
1093                                     |-
1094
1095 174266 052737 000020 0000000 bvs    01N1LUP,00054C3R ;light the init lamp

```

MEMORY TESTS

```

1097      .sdlll MEMORY TESTS
1098      ;+
1099      ;*****
1100      ;
1101      ;      Test the first 2,048 bytes of Program Memory, and
1102      ;      Find a working 8 Kword partition in Program Memory
1103      ;
1104      ;inputs:      PPU Kernel PAR 5 = unit 0 (lower byte), type of boot (upper byte)
1105      ;              PPU Kernel PAR 6 = CSR of TUS0 to use for boot
1106      ;              PPU Kernel PAR 7 = validation word for unit and CSR parameters
1107      ;
1108      ;process: Find an 8 Kword partition in Program Memory that passes a combined
1109      ;          data integrity and addressing test. In addition to this 8 Kword
1110      ;          partition, the first 2048 bytes of Program memory must be working for
1111      ;          use as the POP-11 interrupt and trap vector area.
1112      ;          If the first 2048 bytes do not work, the entire test
1113      ;          is considered to have failed. If the first 2048 bytes are working,
1114      ;          then try to find any 8 Kword chunk of Program memory that works. If the
1115      ;          first 2048 bytes are working, but no contiguous 8 Kword partition can
1116      ;          be found, the test fails.
1117      ;
1118      ;outputs: case 1 - no failure detected
1119      ;          'Init' lamp is on, 'Fault' lamp is off
1120      ;          00 = Base address of tested 8 Kword partition
1121      ;          PPU Kernel PAR's 5,6, and 7 are unchanged
1122      ;          Control is passed to the TUS0 test
1123      ;
1124      ;          case 2 - failure detected (see above for definition of failure)
1125      ;          'Fault' lamp is on
1126      ;          'Init' lamp is on
1127      ;          F-11 is in loop, monitoring the fault lamp
1128      ;          Depressing the fault lamp causes blinking display of fault code.
1129      ;          F-11 must be rebooted to attempt recovery
1130      ;
1131      ;notes:
1132      ;      1) the variable 'PARSIZ' defines the size of the partition to be tested.
1133      ;      2) the variable 'VECSIZ' defines the size of the vector area to test.
1134      ;
1135      ;history of changes to this test:
1136      ;
1137      ;*****
1138      ;-

```


For Internal Use Only

HCSD P.IOC RM BOOT V01-00 MACRO M1200 16-MAY-82 11:16 PAGE 24

..TEST 1 - PROGRAM MEMORY (SWAP BITS)

```

1140                                     .cbit1 ..TEST 1 - Program Memory (Swap Bits)
1141                                     {
1142                                     {
1143                                     {
1144                                     {   Test the 'Swap Boards' and 'Swap Banks' bits in the P.ioc CSR
1145                                     {
1146                                     {inputs: none
1147                                     {
1148                                     {process: 1) set both swap bits
1149                                     {           2) Check that 'swap boards' is set
1150                                     {           3) Clear the 'swap boards' bit
1151                                     {           4) Check that the 'swap banks' bit is still set
1152                                     {           5) Clear the 'swap banks' bit
1153                                     {           6) Check that both swap bits are clear
1154                                     {
1155                                     {outputs: Declare a P.ioc failure if any of the tests fail.
1156                                     {
1157                                     {
1158                                     {
1159 174274 005237 172340                inc    TESTNO        ;set test 0 to 1
1160 174300 012704 000000                mov    @PIOC3R,R4        ;point R4 to P.ioc's CSR
1161 174304 052714 006000                bis    @SWPBD!SWPBK,(R4) ;set both swap bits
1162 174310 032714 004000                bit    @SWPBD,(R4)      ;check the swap board bit
1163 174314 001003                      bne     36              ;br if bit is set
1164 174316 042714 006000 40:          bic    @SWPBD!SWPBK,(R4) ;try to clear both bits
1165 174322 000736                      br      CPUFAL          ;declare a P.ioc failure
1166
1167 174324 042714 004000 36:          bic    @SWPBD,(R4)      ;clear the swap boards bit
1168 174330 032714 006000                bit    @SWPBD!SWPBK,(R4) ;test both bits (one should be set)
1169 174334 001770                      beq     40              ;br if neither bit is set
1170 174336 042714 002000                bic    @SWPBK,(R4)      ;clear the swap banks bit (both clear now)
1171 174342 032714 006000                bit    @SWPBD!SWPBK,(R4) ;check for both bits clear
1172 174346 001363                      bne     40              ;br if either bit failed to clear

```

For Internal Use Only

WDC50 P.IOC RSP BOOT V01-00 MACRO M290 16-MCU-02 11:16 PAGE 25

..TEST 2 - PROGRAM MEMORY (VECTOR AREA)

```

1174      .subttl ..TEST 2 - Program Memory (Vector area)
1175 174350 005237 172940      inc     TESTING      ;set test 0 to 2
1176 174354      MEMTST: j+
1177      ;-----
1178      ;
1179      ;      Test the first 2048 bytes of program memory
1180      ;
1181      ;      If a failure occurs in the first 2048 bytes, then switch
1182      ;      the memory bank that occupies that address range, if
1183      ;      possible, and repeat test. If the memory bank can not be
1184      ;      switched, or if the second bank also fails, declare error.
1185      ;
1186      ;-----
1187      ;
1188 174354 005004      clr     R4      ;address 0 is base address of area to test
1189 174356 012705 003776      mov     @UECSIZ-2,R5      ;R5 gets last address to test
1190 174362 005003      clr     R3      ;R3 gets 0 of failing addresses to store (none)
1191 174364 004737 176570      jar     PC,H0VDM      ;call the memory test
1192 174370 103022      bcc     PARTST      ;br if memory test passed
1193 174372      PARTST: j+
1194      ;-----
1195      ;
1196      ;      if( 'bank swap'=0 ) then ( 'bank swap' gets a 1, retry test )
1197      ;      if( 'bank swap'=1 AND 'board swap'=1 ) then ( declare unrecoverable error
1198      ;
1199      ;      if( 'bank swap'=1 AND 'board swap'=0 ) then ( 'bank swap' gets a 0,
1200      ;      'board swap' gets a 1
1201      ;      retry the test      )
1202      ;
1203      ;      NOTE ON PROGRESSION OF 'SWAP BANK' AND 'SWAP BOARD' SIGNALS:
1204      ;
1205      ;      Pass 1 - 'swap bank' and 'swap board' both zero
1206      ;      Pass 2 - 'swap bank' = 1, 'swap board' = 0
1207      ;      Pass 3 - 'swap bank' = 0, 'swap board' = 1
1208      ;      Pass 4 - 'swap bank' = 1, 'swap board' = 1
1209      ;
1210      ;-----
1211 174372 012706 000000      mov     @KPAR5,SP      ;reset SP
1212 174376 012703 000000      mov     @PIOC5,R3      ;R3 points to P.ioc Control and Status register
1213 174402 032713 002000      bit     @SWBANK,(R3)      ;are banks swapped now?
1214 174406 001003      bne     105      ;br if banks are now swapped
1215 174410 052713 002000      bis     @SWBANK,(R3)      ;assert 'swap banks' signal
1216 174414 000757      br     MEMTST      ;retry memory test on first 2048 bytes
1217      ;
1218 174416 032713 004000      100: bit     @SWBOARD,(R3)      ;are boards swapped now?
1219 174422 001061      bne     BACKEM      ;branch if banks and boards are both swapped already
1220 174424 042713 002000      bic     @SWBANK,(R3)      ; 'swap banks' gets a 0
1221 174430 052713 004000      bis     @SWBOARD,(R3)      ; 'swap boards' gets a 1
1222 174434 000747      br     MEMTST      ;retry memory test

```

..TEST 3 - PROGRAM MEMORY (PARTITION AREA)

```

1224                                .cmt1 ..TEST 3 - Program Memory (Partition area)
1225 174436  PARTST:  j+
1226                                |-----|
1227                                |
1228                                |   Find a working 8 word partition in Program Memory
1229                                |
1230                                |   If no failures are detected, then continue with TUSB test.
1231                                |   If a failure is detected, set the base address of the area
1232                                |   to be tested to the next address following the failure, and
1233                                |   try to find a contiguous 8 word area with no errors.  If no
1234                                |   8 word area can be found, declare a memory test error.
1235                                |
1236                                |-----|
1237                                j-
1238 174436 005237 172340  inc TESTNO ;set test 0 to 3
1239 174442 012706 004000  mov 0UECSIZ,SP ;set stack pointer to top of tested 2048 bytes
1240 174446 012737 174314 000004  mov 0WEIER,4 ;set timeout trap vector
1241 174454 012737 000340 000006  mov 0340,6 ;set trap 4 psw word
1242 174462 012737 174314 000114  mov 0WEIER,00114 ;set parity trap vector
1243 174470 012737 000340 000116  mov 0340,00116
1244 174476 012704 004000  mov 0UECSIZ,R4 ;R4 gets first address to test
1245 174482 012705 043776  mov 0UECSIZ+PARSIZ-2,R5 ;R5 gets last address to test
1246 174486 004737 176370  MEMTY: jvr PC,0EVI00 ;perform the Flipping-Inversions test
1247 174492 103023  bcc MEMDON ;br if memory test passed
1248                                j+
1249                                |-----|
1250                                |
1251                                |   R0 contains the memory address that failed.
1252                                |
1253                                |   Add 2 to R0, then adjust R0 upwards (if necessary) to the
1254                                |   next even 32 word boundary. Start the test over there.
1255                                |
1256                                |   if (last address of next partition is beyond the end of memory)
1257                                |   then ( declare a program memory failure)
1258                                |   else ( test the newly defined partition)
1259                                |
1260                                |-----|
1261                                j-
1262                                |
1263 174514 012706 004000  WEIER: mov 0UECSIZ,SP ;reset SP in case we trapped to here
1264 174520 010004  mov R0,R4 ;move failing address to R4
1265 174522 062704 000002  add 02,R4 ;adjust address past failure
1266 174526 032704 000077  bit 077,R4 ;is address on 32 word boundary?
1267 174532 001404  bcc 100 ;br if it is
1268 174534 042704 000077  bic 077,R4 ;adjust address to 32 word boundary
1269 174540 062704 000100  add 0100,R4
1270 174544 010405 100: mov R4,R5 ;set R5 to starting address
1271 174546 062705 037776  add 0PARSIZ-2,R5 ;R5 now = last address of 8 word partition to test
1272 174552 020527 160000  cmp R5,0160000 ;see if we are into I/O page yet
1273 174556 103003  lhrs 0AC0E0 ;br if we are into I/O page
1274 174560 000732  br MEMTY ;retry memory test on new partition
1275                                |
1276 174562 010400  MEMDON: mov R4,R0 ;R4 gets base address of 8 word partition
1277 174564 000437  br T3BTST ;now go do TUSB test and load

```

For Internal Use Only

MCS50 P.10C RM 0007 U01-00 MICRO M200 16-MCU-02 11:16 PAGE 27

FAULT LAMP ROUTINE

1279			.start	FAULT Lamp Routine
1280			.enable	lab
1281	174566		000001:	jt
1282				
1283				
1284				Declare a memory test error.
1285				Set the fault code register (R0) to 'Memory Module Failure'
1286				
1287				
1288				
1289	174566	012700	000022	mov
1290	174572			000001:
1291				jt
1292				
1293				Handle Fault lamp report for any error. R0 contains failure code
1294				
1295				Clear all lamps except INIT (indicates whether F-11 tests passed)
1296				Light the Fault lamp, then monitor the fault switch. When the Fault
1297				switch is pressed, display a blinking failure code in the lamps.
1298				
1299				
1300				
1301	174572	106427	000340	stop
1302	174576	012701	00000008	mov
1303	174602	042711	000017	bic
1304	174606	052711	000010	bis
1305	174612	032711	004000	000:
1306	174616	001775		beq
1307				jt
1308				
1309				
1310				Display the fault code in the lamps for 1/2 second
1311				
1312				
1313				
1314	174620	042711	000037	000:
1315	174624	010011		mov
1316	174626	012703	000012	PTIME0:
1317	174632	012702	045570	000:
1318	174636	077201		700:
1319	174640	077304		sub
1320				jt
1321				
1322				
1323				Extinguish all the lamps for 1/2 second
1324				
1325				
1326				
1327	174642	042711	000037	bic
1328	174646	012703	000012	PTIME1:
1329	174652	012702	045570	000:
1330	174656	077201		700:
1331	174660	077304		sub
1332	174662	000756		br
1333				.disable

```

1335      .start ..TEST 4 - TUSB Tests
1336      ;
1337      ;=====
1338      ; Find a working TUSB tape drive with media mounted and
1339      ; read the first 8 blocks to the 0K partition just tested.
1340      ;
1341      ;inputs: 00 = Base address of the 0 word partition(supplied by memory test)
1342      ;          00U Kernel PAR 5 = Unit 0 of TUSB(lower byte) and type of boot (upper bit)
1343      ;          00U Kernel PAR 6 = CSR address of TUSB to use for boot (16 bits)
1344      ;          00U Kernel PAR 7 = Validation word for unit and CSR
1345      ;
1346      ;process: Find a working TUSB tape drive that can be used to read in the last P.10C test.
1347      ;          First, a check is made to see if we were passed a specific TUSB CSR and unit to us
1348      ;
1349      ;          The parameters in 00U Kernel PAR's 5, 6 and 7 are checked for validity by testing
1350      ;          the validation word is equal to the XOR of the Unit & CSR words. If the parameters
1351      ;          are valid, the load is attempted via the TUSB specified. If the parameters in the
1352      ;          PAR's are not valid, or if the load attempt fails on the tape specified, the defau
1353      ;          selection sequence (described below) is used.
1354      ;
1355      ;          DEFAULT SELECTION SEQUENCE
1356      ;          Any of the 4 TUSB drives that can be connected to the P.10C can be used as a
1357      ;          boot device. Testing begins with TUSB controller 01, unit 0. If unit 0 does not
1358      ;          respond, unit 1 is tried. If unit 1 also fails, the second TUSB controller is
1359      ;          tested; unit 0 first, then unit 1 if unit 0 fails. Finally, the serial inter-
1360      ;          face to the local terminal is tried if all 4 TUSB drives are unusable for booting.
1361      ;          Both unit 0 and unit 1 are used in this case also.
1362      ;
1363      ;          The test consists of directing the selected TUSB unit to perform its
1364      ;          self test. If the self-test fails, the next unit in sequence is tried.
1365      ;          If the self-test passes, the unit is directed to read its first 8
1366      ;          blocks to the 0 word partition in Program memory. If the first word
1367      ;          of the loaded image is a NOP instruction, control is passed to the
1368      ;          first instruction of the loaded image. If the first instruction is not
1369      ;          a NOP, then the next TUSB unit in sequence is tried.
1370      ;
1371      ;          Failure to find any working TUSB drive with media mounted is a fatal
1372      ;          error, and results in the failure action listed below (see 'outputs').
1373      ;
1374      ;outputs: Case 01 - A working TUSB with bootable media mounted was found:
1375      ;          Parameters passed:
1376      ;          a) Type of boot (warm or cold)
1377      ;          b) Base address of tested 0 word partition
1378      ;          c) Unit number of the TUSB that was used to load
1379      ;          d) CSR address of the TUSB that was used to load
1380      ;
1381      ;          Case 02 - No working TUSB could be found:
1382      ;
1383      ;          State of F-11:
1384      ;          a) 'Int' lamp is lit
1385      ;          b) 'Fault' lamp is lit
1386      ;          c) F-11 is loop monitoring the 'fault lamp'. Depressing
1387      ;             the 'fault lamp' results in a blinking display of the
1388      ;             fault code for 'TUSB Failure'.
1389      ;          d) F-11 must be rebooted to attempt recovery
1390      ;
1391      ;          Probable failures:
1392      ;          a) No HBC50 system cassette is mounted
  
```


..TEST 4 - TUSB TESTS

```

1392 |
1393 |
1394 |
1395 |
1396 |
1397 |
1398 |
1399 |
1400 |
1401 |
1402 |
1403 |
1404 |
1405 |
1406 |
1407 |
1408 |
1409 |

```

b) Faulty TUSB unit (if boot fails on just one unit)
c) Faulty TUSB controller (if boot fails on both units)

Recovery action:
a) Mount an HBC30 system cassette (if none was mounted) and reboot.
b) Switch the HBC30 system cassette to the other TUSB unit.
(if proper media was mounted)
c) Refer to description of TUSB error table in the USEFUL INFORMATION
section of this listing.

History of changes to this test:

=====

1411				.enable job	
1412	174664	005237	172300	test: inc	TESTNO ;set test 0 to 4
1413				if	
1414				-----	
1415				;	
1416				;	Set the stack pointer (R6) to the highest location
1417				;	tested. Note that since we only load 8(10) blocks,
1418				;	we avoid overwriting this area when reading from tape.
1419				;	
1420				-----	
1421				if	
1422	174670	010006		mov	R0,SP ;point stack to lowest address of BK partition
1423	174672	062706	040000	add	0PARS12,SP ;adjust to last address (+2) of BK partition
1424				if	
1425				-----	
1426				;	
1427				;	See if a TUSB unit and CSR was specified for booting.
1428				;	
1429				;	if (unit and CSR are valid) then (try specified unit)
1430				;	else (use default selection sequence)
1431				;	
1432				;	if (specified unit fails) then (use default sequence)
1433				;	
1434				-----	
1435				if	
1436	174676	013704	0000000	mov	0KPAR5,R4 ;R4 gets the unit word (upper bit=type of boot)
1437	174702	042704	100000	bic	0100000,R4 ;clear out type of boot field
1438	174706	013702	0000000	mov	0KPAR6,R2 ;R2 gets the CSR word
1439	174712	074402		xor	R4,R2 ;R2 gets XOR of unit and CSR words
1440	174714	020237	0000000	cmp	R2,0KPAR7 ;is specified unit and CSR valid?
1441	174720	001020		bne	50 ;br if not and use default sequence
1442	174722	042704	177400	bic	0177400,R4 ;clear out upper byte of unit word
1443	174726	013701	0000000	mov	0KPAR6,R1 ;R1 gets CSR address for TUSB subroutine use
1444	174732	020127	160000	cmp	R1,0160000 ;insure CSR address in I/O page
1445	174736	103411		blo	50 ;br if CSR is not in I/O page, use default sequence
1446	174740	020427	000001	cmp	R4,01 ;insure unit 0 is not greater than 1
1447	174744	003006		bgt	50 ;br if unit 0 is greater than 1, use default seq
1448	174746	004737	175226	jsr	PC,T50SST ;Perform sync and self-test on TUSB
1449	174752	103403		bcs	50 ;br if sync or test failed, use default sequence
1450	174754	004737	175362	jsr	PC,READBT ;read boot blocks and check for bootable image
1451	174760	103054		bcc	transfr ;br if read ok, and bootable image loaded
1452					;(else we fall through to default sequence below)

..TEST 4 - TUSB TESTS

```

1454
1455 174762 012746 173074      30:  mov    OCSRLST,-(SP)    ;stack a pointer to the CSR list
1456 174766 012737 000374 000422  mov    OSLERRS-4,TUERRPT ;point to TU 01's error table entry (-4)
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471 174774 005037 000420      100:  clr     TUERR0          ;clear TUSB SW/HW error codes
1472 175000 062737 000004 000422  add     04,TUERRPT      ;update pointer to next error table entry
1473 175006 017601 000000      mov     00(SP),R1       ;R1 gets RCSR address of next controller to use
1474 175012 001434      bcc     T30FAL         ;br if all controllers have been tried
1475 175014 062716 000002      add     02,(SP)        ;update pointer to next CSR for next time around
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486 175020 005004      clr     R4              ;set unit 0 to zero
1487 175022 004737 175226      jsr     PC,T30SST      ;call the sync and self-test routine
1488 175026 103003      bcc     150            ;br if sync and self-test passed
1489 175030 004737 176344      jsr     PC,SAUER0      ;save error codes in table
1490 175034 000757      br      100            ;try the next controller
1491
1492 175036      150:  jr      +
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504 175036 005037 000420      200:  clr     TUERR0          ;clear TUSB HW/SW error codes
1505 175042 004737 175362      jsr     PC,READBT      ;read boot blocks and check for bootable image
1506 175046 103010      bcc     300            ;br if read successful and image is bootable
1507 175050 004737 176344      jsr     PC,SAUER0      ;save error codes in table

```

Select next TUSB controller for bootstrap. The controllers are tried in the following order:

- 1) TUSB 01
- 2) TUSB 02
- 3) Console interface

If (all controllers have been tried)
Then (declare a TUSB failure)

Synchronize with the TUSB and issue self-test command

Set R4 = 0 to indicate unit 0 (not required for sync, but used in the 'READBT' routine called below.)

First try to read a bootable image from unit 0 of the currently selected controller. If we can't load a bootable image from unit 0, try unit 1.

(If the read succeeds, we have read the boot blocks and the TUSB directory into the 0 word partition.)

```

1509
1510
1511
1512
1513
1514
1515
1516
1517 175054 005704
1518 175056 001346
1519 175060 005204
1520 175062 004737 175630
1521 175066 000763
1522
1523
1524
1525
1526
1527
1528
1529
1530 175070 005726
1531 175072 000407
1532
1533
1534
1535 175074 000000
1536 175076 000000
1537 175100 000000
1538 175102 000000
1539
1540
1541
1542 175104
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555 175104 012700 000023
1556 175110 000630
1557

```

```

;+
;-----
;
;   If ( Unit = 1 ) Then ( Try the next controller )
;   Else ( set Unit to 1, try booting from unit 1 )
;-----
;+
;-----
;
;   is unit 0 zero now ?
;   br if not and try next controller
;   set unit to 1
;   pre-spec with controller
;   try reading from unit 1
;-----
;+
;-----
;
;   Remove pointer from stack and continue boot
;-----
;+
;-----
;
;   pop pointer from stack
;   T000FR
;-----
;
;list of TUSB CSR's for booting
CSRLIST: .word 00L103 ;TUSB 01 RCSR address
         .word 00L205 ;TUSB 02 RCSR address
         .word 00L003 ;console RCSR address
         .word 0      ;list terminator
;-----
;
;T00FAL: ;+
;-----
;
;   Declare TUSB error.
;
;   1) Set fault code to 'TUSB failure'
;
;   2) Light the fault lamp
;
;   3) Enter loop to monitor fault lamp
;-----
;+
;-----
;
;   RD gets error code for TUSB failure
;   go to common error reporter
;   .endb lab

```

..TEST 5 - TRANSFER CONTROL TO LOADED IMAGE

```

1559      .cvt11 ..TEST 5 - Transfer Control to Loaded Image
1560
1561      ;
1562      ;
1563      ; The image just loaded to Program Memory is either the 'Init P.IOC' test,
1564      ; or an non-HBC50 bootstrap (e.g. RT-11, REX-11). In either case, the image
1565      ; must be 'messaged' to work properly. In the case of the 'Init P.IOC' test,
1566      ; a one block (512 byte) 'hole' exists between the first and second blocks
1567      ; of the test, due to the presence of a 'Volume ID' block on the TUSO. This
1568      ; 'hole' is collapsed before transferring control to the loaded image. In the
1569      ; case of a 'foreign' (non-HBC50) bootstrap, the image must be moved from
1570      ; the tested 0 hard partition down to address 000000 to work properly.
1571      ;
1572      ; After 'messaging' the loaded image, set up the parameters to be passed and
1573      ; transfer to the image. For the 'Init P.IOC' test, transfer to the base address
1574      ; of the loaded image. For a 'foreign' bootstrap, transfer control to location 0
1575      ;
1576      ;inputs:      All tests passed (implied)
1577      ;              TUSO parameters (CSR, Unit 0)
1578      ;              Base address of working 0 hard partition
1579      ;              Type of boot (warm or cold)
1580      ;              'Init' lamp is lit
1581      ;              'Fault' lamp is off
1582      ;
1583      ;process:      Check the second instruction of the image just loaded.
1584      ;
1585      ;              if ( second instruction = NOP ) then ( Init P.IOC test was loaded )
1586      ;              else ( a foreign bootstrap was loaded )
1587      ;
1588      ;              if ( Init P.IOC test was loaded )
1589      ;              then ( collapse the 1 block hole between first and second blocks,
1590      ;                      transfer to first address loaded )
1591      ;
1592      ;              if ( foreign boot was loaded )
1593      ;              then ( move image down to location 0, transfer to loc 0 )
1594      ;
1595      ;outputs:      See 'inputs' above
1596      ;
1597      ;notes:
1598      ;
1599      ;History of changes to this routine:
1600      ;
1601      ;
1602      ;-

```

```

1604                                     .enabl sub
1605 175112    TIMFR:  ;+
1606                                     -----
1607                                     ;
1608                                     ;   Set up the parameters that specify the TUS0 unit and CSR used for
1609                                     ;   the load just completed. They may have changed if the unit
1610                                     ;   specified for booting was not working.
1611                                     ;
1612                                     -----
1613                                     ;-
1614 175112 005237 172340    inc    TESTNO          ;set test 0 to 5
1615 175116 042737 077777 000000    bic    077777,000KPAR5 ;mask everything except type of boot bit
1616 175124 110437 000000    movb   R4,000KPAR5    ;save unit 0 we booted from
1617 175130 010137 000000    mov    R1,000KPAR6    ;save CSR of TUS0 we booted from
1618                                     ;+
1619                                     -----
1620                                     ;
1621                                     ;   Determine if we loaded the 'Init P.10C' test or a foreign bootstrap
1622                                     ;   if (second instruction of loaded image = NOP)
1623                                     ;   then ( Init P.10C test was loaded)
1624                                     ;   else ( foreign boot was loaded )
1625                                     ;
1626                                     -----
1627                                     ;-
1628 175134 010002    mov    R0,R2          ;R2 gets base address of 8 Kword partition
1629 175136 026227 000002 000200 100:  cmp    2(R2),0000200    ;is second instruction a NOP ?
1630 175144 001017    bne    506          ;br if not, must be a foreign bootstrap
1631                                     ;+
1632                                     -----
1633                                     ;
1634                                     ;   We have loaded the 'Init P.10C' test.
1635                                     ;   The 8 blocks loaded are as follows:
1636                                     ;
1637                                     ;   block 0      first 512 bytes of 'Init P.10c' test
1638                                     ;   block 1      RT-11 volume 10 block
1639                                     ;   block 2      second 512 bytes of 'Init P.10c' test
1640                                     ;   block 3      third 512 bytes of 'Init P.10c' test
1641                                     ;   block 4      fourth 512 bytes of 'Init P.10c' test
1642                                     ;   block 5      fifth 512 bytes of 'Init P.10c' test
1643                                     ;   block 6      first half of first RT-11 directory segment
1644                                     ;   block 7      second half of first RT-11 directory segment
1645                                     ;   We must move blocks 2 thru 7 down one block to collapse the hole.
1646                                     -----
1647                                     ;-
1648 175146 010203    mov    R2,R3          ;R3 gets base address used for load
1649 175150 062703 001000    add    01000,R3    ;R3 now points to 'hole'
1650 175154 010304    mov    R3,R4          ;R4 gets R3
1651 175156 062704 001000    add    01000,R4    ;R4 now points to first address beyond 'hole'
1652 175162 012705 003000    mov    0100512-2+400,R5 ;R5 will count words of data to be moved
1653 175166 012423    200:  mov    (R4)+,(R3)+  ;move a word
1654 175170 077502    sub     R5,200        ;branch until blocks are moved

```


..TEST 5 - TRANSFER CONTROL TO LOADED IMAGE

```

1636                                     ;+
1637                                     |-----|
1638                                     |
1639                                     |   Set up parameters to be passed to the 'INIT P.IOC' test.
1640                                     |
1641                                     |   R0 = Unit 0 of TUSB used to boot, and 'type of boot'
1642                                     |   R1 = CSR address of TUSB used to boot
1643                                     |   R2 = Base address of the tested 8 Hard partition
1644                                     |
1645                                     |-----|
1646                                     ;+
1647 173172 013700 0000000  mov  SUPARS,R0      ;R0 gets unit 0 and type of boot
1648 173176 013701 0000000  mov  SUPARS,R1      ;R1 gets CSR address of TUSB used for boot
1649 173202 000112          jmp  (R2)          ;transfer to the Init P.ioc test
1650
1651                                     ;+
1652                                     |-----|
1653                                     |
1654                                     |   The image just loaded is a foreign (non-HBC50) bootstrap. The image was
1655                                     |   loaded to the 8 Hard partition in Program memory, but is designed to
1656                                     |   execute from locations 000000 thru 000776. So we move the first block
1657                                     |   from the 8 Hard partition, down to address 000000, before jumping
1658                                     |   to address 000000 to begin the bootstrap.
1659                                     |
1660                                     |-----|
1661                                     ;+
1662 175204 010203 300:  mov  R2,R3          ;R3 gets base address of load
1663 175206 003004      clr  R4            ;R4 points to where to start moving data
1664 175210 012705 000400  mov  0400,R5      ;set R5 to count 1 block of data
1665 175214 012324 600:  mov  (R3)+,(R4)+    ;move a word
1666 175216 077502      sub  R3,600        ;branch until block is moved
1667                                     ;+
1668                                     |-----|
1669                                     |
1670                                     |   Now jump to location 0 to continue foreign bootstrap
1671                                     |
1672                                     |-----|
1673                                     ;+
1674 175220 113700 0000000  movb  SUPARS,R0     ;R0 gets unit 0 word
1675 175224 003007      clr  PC            ;jump to 0 to start foreign bootstrap
1676                                     .doubt  ;sb

```



```

1698          .cbl1 Issue TUSB Sync and Self-Test
1699
1700      ;+
1701      ;-----
1702      ;
1703      ; Issue TUSB synchronize and self-test sequence
1704      ;
1705      ;inputs:
1706      ; R0 = Base address of tested partition in Program Memory
1707      ; R1 = CSR address of TUSB to use
1708      ; R2,R3,R5 are SCRATCH
1709      ;
1710      ;outputs:
1711      ; 1) Perform time-out test on TUSB I/O registers.
1712      ; 2) Issue TUSB synchronization sequence.
1713      ; 3) If (sync fails) then (exit with c bit set)
1714      ; 4) Issue TUSB self-test command
1715      ; 5) If (self-test fails) Then (exit with c bit set)
1716      ; Else (exit with c bit clear)
1717      ; NOTE: Loop-back mode can not be used as a test, due to lack of space
1718      ;
1719      ;NOTES:
1720      ; R0,R1 are PRESERVED
1721      ; R2,R3, and R5 are Clobbered
1722      ;
1723      ;-----
1724      ;-
1725
1726      .cbl1 Job
1727 175226 T9008T: ;+
1728      ;-----
1729      ;
1730      ; Check for Non-Existent-Memory trap when accessing the four
1731      ; TUSB I/O registers. If any one of the four addresses results
1732      ; in a NEM trap, exit this routine with C bit set.
1733      ;
1734      ;-----
1735      ;-
1736 175226 013746 000004 mov 004,-(SP) ;save the present contents of the NEM vector
1737 175232 012737 175344 000004 mov 0305,004 ;replace NEM vector with address of 305 below
1738 175240 010162 mov R1,R2 ;R2 gets address of TUSB receiver status register
1739 175242 005722 tst (R2)+ ;test recvr status reg
1740 175244 005722 tst (R2)+ ;test recvr buffer
1741 175246 005722 tst (R2)+ ;test mnttr status
1742 175250 005722 tst (R2)+ ;test mnttr buffer
1743 175252 012637 000004 100: mov (SP)+,004 ;restore NEM vector
1744 175256 103431 bcs 200 ;br if we had a timeout
1745      ;+
1746      ;-----
1747      ;
1748      ; Perform TUSB synchronization sequence
1749      ;
1750      ;-----
1751      ;-
1752 175260 004737 175630 jbr PC,T900YC ;sync with TUSB
1753 175264 103426 bcs 200 ;br if sync failed
1754      ;-
  
```

100

```

1005                                     .subtl Read TUSB Boot Blocks to Program Memory
1006
1007                                     ;+
1008                                     ;-----
1009                                     ;
1010                                     ; Read TUSB boot blocks to Program Memory and test for Bootable image
1011                                     ;
1012                                     ;inputs:
1013                                     ; R0 = Base address of tested Program Memory partition
1014                                     ; R1 = CBR address of TUSB to be used
1015                                     ; R2 = SCRATCH
1016                                     ; R3 = SCRATCH
1017                                     ; R4 = Unit # of TUSB to be used
1018                                     ; R5 = SCRATCH
1019                                     ;
1020                                     ;process:
1021                                     ; Issue a TUSB read command for 0 blocks, load the data beginning at
1022                                     ; the address in R0.
1023                                     ; NOTE: Of the 0 blocks read, 5 are boot blocks, 1 is volume ID block,
1024                                     ;       2 are directory blocks.
1025                                     ;
1026                                     ;outputs:
1027                                     ; C clear if read is successful (no errors and first instruction = NOP)
1028                                     ; C set if read fails or first instruction is not a NOP
1029                                     ; R0,R1, and R4 are PRESERVED
1030                                     ; R2,R3, and R5 are Clobbered
1031                                     ;
1032                                     ;-----
1033                                     ;-
1034 175362  READY: ;+
1035                                     ;-----
1036                                     ;
1037                                     ; call the TUSB loader
1038                                     ;
1039                                     ;-----
1040                                     ;-
1041 175362 010046 mov R0,-(SP) ;save base address of partition on stack
1042 175364 010005 mov R0,R5 ;construct packet buffer ptr in R5
1043 175366 062705 037400 add 0700512-256.,R5
1044 175372 010002 mov R0,R2 ;R2 gets starting address for load
1045 175374 010400 mov R4,R0 ;R0 gets unit # of TUSB
1046 175376 010303 mov R5,R3 ;R3 gets pointer to packet buffer
1047 175400 012704 000010 mov 0100612,R4 ;R4 gets # of blocks to load
1048 175404 003005 clr R5 ;R5 gets block # of 0
1049 175406 004737 175442 jnr PC,T500EAD ;call the read subroutine
1050 175412 010004 mov R0,R4 ;restore unit # to R0
1051 175414 012600 mov (SP)+,R0 ;restore base address in R0
1052 175416 103410 bcs 300 ;br if load failed (device error)

```

READ THIS HOT BLICKS TO PROOF HISTORY

Check for a bootable image just loaded. A bootable image contains a POP-11 NOP instruction in the first word loaded.

```

j-      (R0),0000200      ;is first instruction of image loaded a NBP ?
cmp      200              ;or if yes, we have a bootable image
beq      STUFFY,TIMER     ;remember that we did not find a bootable image
movb     (PC)+,PC         ;set c bit, skip next instruction
cmp      PC               ;clear c bit
clc      PC               ;return
rts

```

```

1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096 175442
1097
1098
1099
1900
1901
1902
1903 175442 010046
1904 175444 010246
1905 175446 010346
1906
1907
1908
1909
1910
1911
1912
1913 175450 112723 000002
1914 175454 112723 000012
1915 175460 112723 000002
1916 175464 103023
1917 175466 110023
1918 175470 103023
1919 175472 103023
1920 175474 103023
1921 175476 012700 000011
1922 175502 006304
1923 175504 077002
1924 175506 005704
1925 175510 001402

```

```

;-----
;      TUSB Read Routine
;-----
;
;      Read from a TUSB to Program Memory
;
;inputs:
;      R0 = TUSB Unit #
;      R1 = TUSB ACSR address
;      R2 = Starting address for load
;      R3 = Address of a packet buffer
;      R4 = # of blocks to read (load)
;      R5 = Starting block for read
;
;process:
;      Construct a read command packet and send it to the TUSB. Load the
;      data read from the tape to Program Memory, and check for a successful
;      read. condition c bit accordingly.
;
;outputs:
;      R0,R1,R2, and R3 are PRESERVED
;      R3,R4, and R5 are Clobbered
;      C bit is clear is read and image check both succeed.
;      C bit set if read fails, or if image is not bootable.
;-----
;
TUSBREAD: ;+
;-----
;      Preserve R0,R1,R2, and R3 on the stack
;-----
;
;-----
;      Construct a read command in the packet buffer
;-----
;
;-----
movb    0000,(R3)+    ;flag byte = 'command'
movb    010.,(R3)+    ;byte count = 10
movb    00EAD,(R3)+   ;op-code = 'read'
clrb    (R3)+         ;modifier = 0
movb    R0,(R3)+      ;insert unit #
clrb    (R3)+         ;switches = 0
clrb    (R3)+         ;seq 0 (low) = 0
clrb    (R3)+         ;seq 0 (high) = 0
mov     R5,R0         ;use R0 to count 9 shifts
100:    shl    R4         ;shift block count 9 times to multiply by 1000(0)
        sub    R0,100    ;(block count times 1000 = byte count)
        tst    R4         ;insure a non-zero number of bytes to load
        beq    400       ;take error out if byte count is zero

```


TUBS READ ROUTINE

```

1926 175512 110423      movb    R4,(R3)+      ;save byte count (low)
1927 175514 000304      movb    R4              ;reverse bytes in R4
1928 175516 110423      movb    R4,(R3)+      ;save byte count (high)
1929 175520 110523      movb    R5,(R3)+      ;save block 0 (low)
1930 175522 000305      movb    R5              ;reverse bytes in R5
1931 175524 110523      movb    R5,(R3)+      ;save block 0 (high)
1932                      ;+
1933                      ;-----
1934                      ;
1935                      ;      Send the read command packet to the TUBS
1936                      ;
1937                      ;-----
1938                      ;-
1939 175526 011603      mov     (SP),R3      ;point R3 to packet just constructed
1940 175530 004737 176136  jbr     PC,SENDPK      ;call the send packet routine
1941 175534 103430      bcs     406      ;br if send packet failed
1942                      ;+
1943                      ;-----
1944                      ;
1945                      ;      Accept data packets from TUBS
1946                      ;
1947                      ;-----
1948                      ;-
1949 175536 016602 000002 206:  mov     2(SP),R2      ;R2 gets starting address for load
1950 175542 011603      mov     (SP),R3      ;point R3 to a packet buffer
1951 175544 004737 176242  jbr     PC,RECVPK      ;call receive packet to input data and command messages
1952 175550 103422      bcs     406      ;br if reception failed
1953                      ;+
1954                      ;-----
1955                      ;
1956                      ;      If (last packet = data packet) then (receive another packet)
1957                      ;      If (last packet = end packet) then (check for success code)
1958                      ;      If (last packet not data or end packet) then (error)
1959                      ;
1960                      ;-----
1961                      ;-
1962 175552 011603      mov     (SP),R3      ;point R3 to buffer where last packet was inputted
1963 175554 121327 000001      cmovb   (R3),00ATA      ;check for data ;ack:
1964 175560 001770      beq     206      ;br if data packet, go get another packet until 'end'
1965 175562 126327 000002 000100      cmovb   2(R3),0ENDP      ;check for end packet
1966 175570 001404      beq     236      ;br if end packet
1967 175572 112737 000011 000420      movb    0TUPROT,TUBIER      ;remember we got an unknown packet type from the TU
1968 175600 000406      br       406
1969                      ;
1970 175602 105763 000003 236:  tstb    3(R3)      ;check for success code in end packet
1971                      ;clc
1972 175606 100004      bpl     306      ;br if end packet indicates success
1973 175610 116337 000003 000421      movb    3(R3),TUBIER      ;save error code from end packet
1974 175616 000261      sec     406:
1975 175620 012603 306:  mov     (SP)+,R3      ;restore packet buffer pointer
1976 175622 012602      mov     (SP)+,R2      ;restore load starting address
1977 175624 012600      mov     (SP)+,R0      ;restore unit 0
1978 175626 000207      rts     PC              ;return

```



```

1980 .subttl Synchronize with a TUSB
1981 ;
1982 ;-----
1983 ; Subroutine to synchronize with a TUSB
1984 ;
1985 ;inputs:
1986 ; R1 = RCSR address of TUSB to synchronize
1987 ; R2 = SCRATCH
1988 ; R3 = SCRATCH
1989 ;process:
1990 ; Send a 'break' to the TUSB for 1 character time. Then send 2 'Init' characters
1991 ; Wait up to 2 seconds for the TUSB to respond with a 'Continue'
1992 ;outputs:
1993 ; R1 = PRESERVED
1994 ; R2 = Clobbered
1995 ; R3 = Clobbered
1996 ;
1997 ;-----

```

```

1998 ;
1999 ;
2000 175630 .enable lsb
2001 TUSBYC: ;
2002 ;-----
2003 ; Synchronize the F-11 with the TUSB. The F-11 sends a 'break'
2004 ; character to the TUSB for one character time, then the F-11
2005 ; sends two TUSB 'init' commands and waits 40 seconds for the
2006 ; TUSB to respond with a 'Continue' message. If the F-11 does
2007 ; not respond correctly, exit this routine with the c bit set,
2008 ; otherwise exit with c bit clear
2009 ;
2010 ;-----
2011 ;
2012 175630 012761 000001 000004 mov 00(break,4(R1)) ;set break bit in xmitter status register
2013 175636 005082 clr R2 ;set R2 to the 'null' character
2014 175640 004737 176004 jsr PC,sendat ;send R2 and wait for xmitter ready
2015 175644 103451 bcs 300 ;br if time out waiting for 'xmit ready'
2016 175646 004737 176004 jsr PC,sendat ;send another null (times length of break)
2017 175652 103446 bcs 300 ;br if time out waiting for 'xmit ready'
2018 175654 105761 000004 100: tstb 4(R1) ;wait for xmitter ready
2019 175660 100375 bpl 100 ;when ready, indicates first null done sending
2020 175662 005061 000004 clr 4(R1) ;clear the break bit
2021 175666 012782 000004 mov 0INIT,R2 ;set R2 to tusb 'INIT' character
2022 175672 004737 176004 jsr PC,sendat ;send char in R2 and wait for 'Xmit ready'
2023 175676 103434 bcs 300 ;br if time out waiting for 'xmit ready'
2024 175700 004737 176004 jsr PC,sendat ;send another INIT
2025 175704 103431 bcs 300 ;br if time out waiting for 'xmit ready'
2026 175706 105761 000004 200: tstb 4(R1) ;wait for xmitter ready to set
2027 175712 100375 bpl 200 ;when ready, indicates first 'cont' done sending
2028 175714 026161 000002 000002 cmp 2(R1),2(R1) ;read rbuf twice to clear out any junk characters

```

For Internal Use Only

HBC30 P.10C RM BOOT V01-00 MACRO M200 16-MCU-02 11:16 PAGE 40

SYNCHRONIZE WITH A TUB

```
2030                                     ;+
2031                                     |-----|
2032                                     |
2033                                     |   Allow up to 2 seconds for TU to respond before
2034                                     |   rejecting this controller
2035                                     |
2036                                     |-----|
2037                                     |~
2038 175722 012703 000050      PTIME2: mov    040.,R3      ;set up for 40 delays of 50 MSEC each (=2 seconds)
2039 175726 012702 015034      210:  mov    06604.,R2      ;R2 gets timing constant
2040 175732 105711      220:  tstb    (R1)      ;get a character yet?      3.14 usec
2041 175734 100403      ;br if we did      1.72 usec
2042 175736 077203      ;br until R2 = 0      2.62 usec (total 7.48)
2043 175740 077306      ;do delay 40 times
2044 175742 000412      br      300      ;take error exit
2045
2046 175744 016102 000002      230:  mov    2(R1),R2      ;get received character to R2
2047 175750 100004      ;br if no UART errors on received character
2048 175752 112737 000010 000420      movb    0TUNCER,TUNCER ;remember we got a UART error
2049 175760 000406      br      350      ;take error exit
2050
2051 175762 120227 000020      240:  cmph    R2,0CONT      ;see if 'TUBS CONTINUE' was received
2052 175766 001404      ;br if continue received
2053 175770 112737 000002 000420      300:  movb    0TUSYNC,TUNCER ;remember we had a sync failure
2054 175776 022707      330:  cmp     (PC)+,PC      ;set c bit, skip next instruction
2055 176000 000201      400:  clc      ;clear c bit
2056 176002 000207      rts     PC      ;return
2057      .endb    lab
```

```

2059          .sbt11 Send Character to TUSB
2060
2061      ;+
2062      ;-----
2063      ;
2064      ;   Send character in R2 to the TUSB specified by address in R1.
2065      ;   Wait up to 1/2 second for 'Xmit ready' to set before declaring
2066      ;   a timeout.
2067      ;
2068      ;inputs:
2069      ;   R1 = Address of a TUSB Receiver Status Register
2070      ;   R2 = Character to be sent
2071      ;   SP -> Return address
2072      ;
2073      ;pretest:
2074      ;   Wait up to 1/2 second for the 'Xmit Ready' flag to set
2075      ;   Move R2 to the TUSB 'Xmit Buffer' after xmit ready sets
2076      ;
2077      ;outputs:
2078      ;   C bit is clear (0) if the transmit was successful
2079      ;   C bit set (1) if the transmit ready flag does not set in time
2080      ;   R2 = undefined
2081      ;
2082      ;-----
2083      ;-
2084      .enabl lab
2085      ;-----
2086      ;+
2087      ;-----
2088      ;   Make sure 'Transmit Ready' is set before transmitting the
2089      ;   character in R2. If 'Transmit Ready' does not set within
2090      ;   the timeout period, declare an error. (exit c set)
2091      ;
2092      ;-----
2093      ;-
2094      176004 010246      mov     R2,-(SP)      ;save character to send on stack
2095      176006 005002      clr     R2          ;clear R2 for use as a counter
2096      176010 105761 000004 100:  tstb    4(R1)      ;test the 'Transmit Ready' bit
2097      176014 100407      bmi     206          ;br if 'Transmit Ready' is set
2098      176016 005202      inc     R2          ;add one to counter
2099      176020 001373      bne     100          ;br if counter is not zero
2100      176022 112737 000006 000420  movb   0TUBRTO,TUBRER ;remember we had a transmit timeout
2101      176030 000261      sec          ;set c bit to indicate time-out
2102      176032 000405      br      300          ;exit with c set
2103      176034      200:  ;+
2104      ;-----
2105      ;
2106      ;   Put the character in R2 into the 'Transmit Buffer'
2107      ;
2108      ;-----
2109      ;-
2110      176034 042716 177400  bic     0177000,(SP) ;mask out upper 8 bits
2111      176040 011661 000006  mov     (SP),6(R1) ; 'Transmit Buffer' (- character to send
2112      176044 000241      cld          ;
2113      176046 012602 300:  mov     (SP)+,R2      ;restore character to R2
2114      176050 000207      rts     PC          ;return
2115      .dsabl lab

```

RECEIVE A CHARACTER FROM A TUS0

```

2117                                     .sbt11 Receive a Character from a TUS0
2118
2119                                     ;+
2120                                     ;=====
2121                                     ;
2122                                     ;   Receive one character from a TUS0
2123                                     ;
2124                                     ;inputs:
2125                                     ;   R1 = address of TUS0 Receiver Status Register
2126                                     ;   R2 = SCRATCH
2127                                     ;
2128                                     ;process:
2129                                     ;   Wait up to 40 seconds for a character to arrive. If a character
2130                                     ;   arrives within the time allowed, put character into R2
2131                                     ;
2132                                     ;outputs:
2133                                     ;   C bit is clear if the reception was successful. R2 = character
2134                                     ;   C bit is set if the reception fails, R2 undefined.
2135                                     ;
2136                                     ;=====
2137                                     ;-
2138
2139                                     .enab1 1sb
2140 176052  recmat: ;+
2141                                     ;-----
2142                                     ;
2143                                     ;   Create a one word counter on the stack to time the arrival
2144                                     ;   of a character, then begin waiting for the character to arrive.
2145                                     ;   (wait up to 40 seconds before declaring timeout)
2146                                     ;
2147                                     ;-----
2148                                     ;-
2149 176052 012746 000110  PTIME3: mov 0750THD,-(SP) ;put secondary timer count on stack
2150 176056 005002          clr R2 ;use R2 as primary timer
2151 176060 105711 100:  test (R1) ;test 'Receiver Done' bit (bit 7)
2152 176062 100412          bms 200 ;br if 'Receiver Done' is set (byte is neg)
2153 176064 005202          inc R2 ;add one to counter
2154 176066 001374          bne 100 ;br if counter is not zero
2155 176070 005316          dec (SP) ;subtract one from secondary timer
2156 176072 001372          bne 100 ;br if secondary is not zero yet
2157 176074 112737 000007 000420  movb 07URCTD,TUMER ;remember we had a receiver timeout
2158 176102 005726 130:  test (SP)+ ;remove counter word from stack
2159 176104 000261          sec ;set c bit to indicate error
2160 176106 000412          br 300 ;exit with c set
2161
2162 176110 200:  ;+
2163                                     ;-----
2164                                     ;
2165                                     ;   Read the character received to R2 and check for errors
2166                                     ;
2167                                     ;-----
2168                                     ;-
2169 176110 016102 000002  mov 2(R1),R2 ;R2 (- contents of 'Receiver Buffer'
2170 176114 100004          bpl 230 ;br if no error set in RBUF
2171 176116 112737 000010 000420  movb 07URCTD,TUMER ;remember we had a UART error
2172 176124 000766          br 130 ;take error exit
2173

```

For Internal Use Only
 MRC50 P.10C ROM BOOT V01-00 MACRO M1200 16-MCU-82 11:16 PAGE 42-1
 RECEIVE A CHARACTER FROM A TUCB

2174 176126 042782 177400	236:	bic	0177400,82	mask character received to 8 bits
2175		if		
2176		;		
2177		;		
2178		;	Return with c bit set (error) or clear (success)	
2179		;		
2180		;		
2181		;		
2182 176132 005726		ist	(SP)+	remove counter word from stack
2183		jcl		
2184 176134 000207	306:	ris	PC	return
2185		.deabi	lsh	

110

For Internal Use Only

HDC50 P.10C RMH 0007 V01-00 MACRO M200 16-MRU-82 11:16 PAGE 44

SEND A TUSO COMMAND PACKET

2244				;	
2245				;	
2246				;	
2247				;	include word in checksum
2248				;	
2249				;	
2250				;	
2251				;	
2252	176150	061666	000004	add	(SP),4(SP) ;add word to checksum
2253	176154	003366	000004	adc	4(SP) ;include carry-out if C bit is set
2254					
2255				;	
2256				;	
2257				;	
2258				;	Send the lower byte of the word on top of stack
2259				;	
2260				;	
2261				;	
2262				;	
2263	176160	011602		mov	(SP),R2 ;R2 gets the word being transmitted
2264	176162	004737	176004	jmp	PC,adwst ;send R2 to transmitter buffer
2265	176166	103422		bcs	1000 ;br if transmission failed
2266					
2267				;	
2268				;	
2269				;	
2270				;	Send the upper byte of the word on top of stack
2271				;	
2272				;	
2273				;	
2274				;	
2275	176170	011602		mov	(SP),R2 ;R2 gets word to send
2276	176172	000302		swab	R2 ;swap bytes so high byte in lower byte of R2
2277	176174	004737	176004	jmp	PC,adwst ;send the lower byte of R2 to xmitter buffer
2278	176200	103415		bcs	1000 ;br if transmission failed
2279					
2280				;	
2281				;	
2282				;	
2283				;	Check to see if it is time to send the checksum word
2284				;	
2285				;	
2286				;	
2287				;	
2288	176202	005366	000002	dec	2(SP) ;subtract 2 from transmission word counter
2289	176206	003357		bgt	100 ;br if another word to transmit

MICRO P.10C RM BOOT V01-00 MICRO P1200 16-MAY-82 11:16 PAGE 45
 READ A TUBE COVERING PACKET

112

```

2330 .sbtll Receive a Command or Data Packet from the TUSB
2331
2332 *****
2333
2334 Receive a Command or Data Packet from a TUSB
2335
2336 inputs:
2337 R1 = address of TUSB Receiver Status Register
2338 R3 = pointer to a packet buffer
2339 R2 = pointer to data storage area (only used for incoming data packets)
2340
2341
2342 if (first byte = 'Data Packet')
2343 then ( input data bytes to memory )
2344 else ( Input 'command packet' to the packet buffer )
2345
2346 Compute checksum
2347
2348 outputs:
2349 R1 = UNCHANGED
2350 R2 = pointer to next data storage location
2351 R3 = UNDEFINED
2352 R4 and R5 are PRESERVED
2353
2354 C bit clear if reception was successful
2355 C bit set if reception failed
2356
2357 NOTES:
2358 1) command packets are just inputted to the packet buffer, and the
2359 checksum is verified before returning to the calling process.
2360
2361 2) The first two bytes of a 'Data' packet are inputted to the packet
2362 buffer, but the data is stored on the fly
2363
2364 3) This routine will wait a maximum of 40 seconds between characters
2365 sent from the TUSB. Time-out while waiting for 'Receiver Done' to
2366 set causes this routine to exit with the C bit set (error indication).
2367
2368 *****
2369
2370 .mahl lsb
2371 176242
2372
2373
2374 Preserve some registers, and create some temporary
2375 storage locations on the stack
2376
2377
2378
2379
2380 176242 010046 mov R0,-(SP) ;preserve R0
2381 176244 010546 mov R5,-(SP) ;preserve R5
2382 176246 005046 clr -(SP) ;create a checksum word
2383 176250 005046 clr -(SP) ;create a message word counter
2384 176252 005046 clr -(SP) ;create a temporary for the receiver process
2385 176254 010200 mov R2,R0 ;put the data storage pointer in R0

```

RECEIVE A CLIPPING OR DATA PACKET FROM THE TUDS

```

NOTE: CURRENT STACK PICTURE

+10    return address
+8     saved R3
+6     saved R0
+4     checksum word
+2     message word counter
SP --> receiver temporary storage word


input a 'low' byte


jcr PC,rcvnet      ;receive a character in R2
bcs 400            ;br if reception error
mov R2,(SP)        ;save inputted byte on stack


input a 'high' byte


jcr PC,rcvnet      ;receive a character in R2
bcs 400            ;br if reception failed
swab (SP)          ;swap bytes on stack
add R2,(SP)        ;include byte just inputted
swab (SP)          ;put bytes in proper order


check for a 'command' or 'data' packet by looking
at the first byte received


mov (SP),(R3)+     ;store first two bytes of packet in buffer
mov (SP),4(SP)     ;include first 2 bytes in checksum
cmpb (SP),0data    ;check for a data packet code
beq 200             ;br if incoming packet is a 'data' packet

```

For Internal Use Only

MSC90 P.10C ROM BOOT V01-00 MACRO M1200 16-MAY-02 11:16 PAGE 49

RECEIVE A COMMAND OR DATA PACKET FROM THE TUSB

2442				;	
2443				;	
2444				;	
2445				;	receive the rest of a 'command' packet, and store it
2446				;	in the packet buffer. Receiver 8 more bytes of command,
2447				;	and 2 bytes of checksum
2448				;	
2449				;	
2450				;	
2451				;	
2452	176316	005266	000002	inc	2(SP) ; bump word counter
2453					
2454	176322		1001	;	
2455				;	
2456				;	
2457				;	get a 'low' byte
2458				;	
2459				;	
2460				;	
2461				;	
2462	176322	004737	176052	jsr	PC,rcvdat ; receive a character in R2
2463	176326	103477		bcs	400 ; br if reception failed
2464	176330	010216		mov	R2,(SP) ; store low byte on stack
2465					
2466				;	
2467				;	
2468				;	
2469				;	get a 'high' byte
2470				;	
2471				;	
2472				;	
2473				;	
2474	176332	004737	176052	jsr	PC,rcvdat ; receive a character in R2
2475	176336	103473		bcs	400 ; br if reception failed
2476	176340	000316		sub	(SP) ; bump bytes on stack
2477	176342	060216		and	R2,(SP) ; include high byte just received
2478	176344	000316		sub	(SP) ; put bytes in proper order
2479					
2480				;	
2481				;	
2482				;	
2483				;	include word just received in checksum, add one to the
2484				;	word counter, and store word to packet buffer
2485				;	
2486				;	
2487				;	
2488				;	
2489	176346	061666	000004	add	(SP),4(SP) ; add word to checksum
2490	176352	005566	000004	adc	4(SP) ; include possible carry in checksum
2491	176356	005266	000002	inc	2(SP) ; add one to word counter
2492	176362	011623		mov	(SP),(R3)+ ; store word to packet buffer

116

2551 176452	256:	{+	
2552			
2553			
2554			Data Packet is incoming. Two bytes received so far are
2555			on the top of the stack. low byte is 'data packet' code,
2556			high byte is the byte count for the data arriving next.
2557			Put byte count in lower byte, and clear upper byte.
2558			
2559			
2560			
2561			
2562 176452 042716 000377		bic	0377,(SP) ;clear out 'data packet' code
2563 176456 000316		mov	(SP) ;put byte count into lower byte
2564 176460 011605		mov	(SP),R5 ;put byte count into R5
2565			
2566 176462	306:	{+	
2567			
2568			
2569			Get a 'low' byte
2570			
2571			
2572			
2573			
2574 176462 004737 176052		jbr	PC,recvat ;receive a character in R2
2575 176466 103417		bcs	400 ;br if reception failed
2576 176470 010216		mov	R2,(SP) ;save low byte
2577			
2578		{+	
2579			
2580			
2581			Get a 'high' byte
2582			
2583			
2584			
2585			
2586 176472 004737 176052		jbr	PC,recvat ;receive a character in R2
2587 176476 103413		bcs	400 ;br if reception failed
2588 176500 000302		mov	R2 ;move byte to proper position
2589 176502 061602		add	(SP),R2 ;include low byte
2590			
2591		{+	
2592			
2593			
2594			include word in checksum
2595			
2596			
2597			
2598			
2599 176504 060266 000004		add	R2,4(SP) ;add inputted word to computed checksum
2600 176510 005566 000004		adc	4(SP) ;include possible carry in checksum

For Internal Use Only

HBC50 P.10C ROM BOOT V01-00 MICRO M200 16-MAY-02 11:16 PAGE 51

RECEIVE A COMMAND OR DATA PACKET FROM THE TUSB

2602			{+	
2603				
2604				
2605				Store inputted word via the pointer in R0
2606				
2607				
2608				
2609				
2610	176314	010220		mov R2,(R0)+
2611				
2612			{+	
2613				
2614				
2615				test for byte count exhausted. If not exhausted, get
2616				another word, otherwise input checksum word
2617				
2618				
2619				
2620				
2621	176316	162705 000002		sub R2,R5 ;byte count gets byte count minus 2
2622	176322	003357		bgt 300 ;br if another word to input
2623	176324	000723		br 150 ;go get checksum
2624				
2625	176326		400:	{+
2626				
2627				
2628				error, set c bit and exit
2629				
2630				
2631				
2632				
2633	176326	000261		sec ;set c bit to indicate error
2634				
2635	176330		500:	{+
2636				
2637				
2638				clean up stack and exit
2639				
2640				
2641				
2642				
2643	176330	010002		mov R0,R2 ;restore the data pointer to R2
2644	176332	012626		mov (SP)+,(SP)+ ;pop off two words (leaving c bit unaffected)
2645	176334	012603		mov (SP)+,R3 ;pop off another word (leaving c bit unaffected)
2646	176336	012605		mov (SP)+,R5 ;restore R5
2647	176340	012600		mov (SP)+,R0 ;restore R0
2648	176342	000207		ret PC ;return
2649				End of sub

```

2652                                     .cblt Routine to save TUSB error reasons
2653                                     ;+
2654                                     ;=====
2655                                     ;
2656                                     ; Save TUSB error reasons
2657                                     ;
2658                                     ;inputs: 'TUEPT' = points to error word for unit 0 of controller we just tried
2659                                     ;         to boot from.
2660                                     ;         R4 = unit 0 we tried to boot from (0 or 1)
2661                                     ;         'TUSER' = error reason generated by ROM software
2662                                     ;         'TUSER' = error code from TUSB end packet (0 if none)
2663                                     ;
2664                                     ;process: 1) select error word to use.
2665                                     ;         IF ( R4 = 0 )
2666                                     ;             THEN ( error word = contents of TUEPT )
2667                                     ;             ELSE ( error word = (contents of TUEPT) + 2)
2668                                     ;         2) Upper byte of error word gets low byte of TUSER
2669                                     ;         Lower byte of error word gets low byte of TUSER
2670                                     ;
2671                                     ;outputs: All registers are preserved.
2672                                     ;
2673                                     ;=====
2674                                     ;-
2675                                     .enabl 1ab
2676 176344 010046 SHVER: mov R0,-(SP)           ;preserve R0
2677 176346 013700 000422      mov TUEPT,R0      ;R0 gets pointer to error word for unit 0
2678 176352 005704           tst R4              ;is unit = 0 ?
2679 176354 001401           bnc 100             ;br if yes
2680 176356 005720           tst (R0)+          ;add 2 to R0
2681 176360 013710 000420 100: mov TUSER,(R0)      ;save 104/34 error codes in table
2682 176364 012600           mov (SP)+,R0      ;restore R0
2683 176366 000207           rts PC              ;return
2684                                     .enabl 1ab

```

```

2686      .sbt11 Moving Inversions Test Description
2687      ;+
2688      ;~~~~~
2689      ;      Modified Moving Inversions ROM Test for PDP-11's
2690      ;
2691      ;TEST STRATEGY: Starting with the memory cleared to zeroes, make 'n' passes (*)
2692      ;      from lowest address to highest address. On each pass, every location is
2693      ;      read to verify that the last pattern was stored correctly, then the data
2694      ;      in the location is rewritten to contain an additional 'one'. This is
2695      ;      continued until each location contains all ones. Now that the memory
2696      ;      contains all 'ones', make 'n' passes thru the memory from highest address
2697      ;      to lowest, this time read each location and verify that the previous
2698      ;      pattern is correct, replace a single '1' with a '0', and reread each
2699      ;      location to insure that the new pattern was stored. When finished with 16
2700      ;      passes, the memory contains all zeroes again.
2701      ;
2702      ; (*) 'n' = number of binary bits per location
2703      ;
2704      ;Test Steps:
2705      ;      1) Clear memory to all 'zeroes'.
2706      ;      2) Starting at the LOWEST address, and working UP to the highest, read
2707      ;      a location and verify that the previous data was stored, replace
2708      ;      the location's data with a single additional 'one' bit, and reread
2709      ;      the location to insure that the new pattern was stored correctly.
2710      ;      This sequence is repeated until, on the final pass, the memory
2711      ;      contains all ones. (16 passes thru the memory).
2712      ;      3) Starting now at the HIGHEST memory address and working DOWN, read
2713      ;      each location and verify that the previous pattern is read cor-
2714      ;      rectly, rewrite the location's data to contain a single additional
2715      ;      'zero' bit, then reread the location to insure that the new pattern
2716      ;      was stored correctly. Continue until the memory contains all zeroes.
2717      ;      (After the 16th pass.)
2718      ;      4) End of test.
2719      ;
2720      ;FAILURE INDICATIONS: On each of the steps described above as a 'check', failure
2721      ;      to find the expected value is an error.
2722      ;
2723      ;~~~~~
2724      ;-
  
```

```

2726                                     .sbt1) Moving Inversions Test
2727
2728                                     ;+
2729                                     ;*****
2730                                     ;INPUTS:
2731                                     ;   R0,R1,R2,R3 are SCRATCH
2732                                     ;   R4 = Base address of the section of memory under test
2733                                     ;   R5 = Last address of the section of memory under test
2734                                     ;   R6 = Stack Pointer. (WARNING: do not push more than 2 words)
2735                                     ;
2736                                     ;OUTPUTS:
2737                                     ;   C Bit set if test FAILED
2738                                     ;       R0 = failing address
2739                                     ;       R4,R5 unchanged
2740                                     ;   C bit clear if test PASSED
2741                                     ;       R4,R5 unchanged
2742                                     ;       R0 thru R3 undefined
2743                                     ;
2744                                     ;REGISTER USAGE:
2745                                     ;   R0 = Address under test
2746                                     ;   R1 = Data Pattern to check for
2747                                     ;   R2 = Data Pattern to store
2748                                     ;   R4 = Base Address of memory under test
2749                                     ;   R5 = Last Address of memory under test
2750                                     ;   R6 = Stack pointer (WARNING: do not push more than 2 words)
2751                                     ;
2752                                     ;*****
2753                                     ;-
2754 176570 MOVINW: ;+
2755                                     ;*****
2756                                     ;   STEP 1: Clear all memory locations to zeroes.
2757                                     ;*****
2758                                     ;-
2759
2760 176570 010400 MOV    R4,R0          ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
2761 176572 005020 100: CLR    (R0)+        ;CLEAR A LOCATION(WORD), BUMP ADDRESS BY 2
2762 176574 020005 CXP    R0,R5          ;CHECK FOR THE LAST ADDRESS CLEARED
2763 176576 101775 BLOS   100          ;BR IF MORE TO CLEAR
2764                                     ;+
2765                                     ;*****
2766                                     ;   STEP 2: Starting at the LOWEST address, and working UP to the highest,
2767                                     ;   read each location and verify that the previous pattern is read
2768                                     ;   correctly, rewrite the location's data to contain a single
2769                                     ;   additional one, then reread the location to verify that the new
2770                                     ;   pattern was stored correctly. This sequence is repeated 16
2771                                     ;   times, until the memory contains all 'ones'.
2772                                     ;*****
2773                                     ;-
2774
2775 176600 005002 CLR    R2          ;INITIALIZE 'PATTERN TO STORE'
2776 176602 010201 200: MOV    R2,R1        ;REPLACE THE LAST PATTERN WITH THE NEXT TO CHECK FOR
2777 176604 006302 ASL    R2          ;SET ONE MORE 'ONE' BIT IN R2(PATTERN TO STORE)
2778 176606 005202 INC    R2          ;(USING 'INC' VS. 'DEC' SAVES TIME)
2779 176610 010400 MOV    R4,R0          ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
2780 176612 020110 300: CXP    R1,(R0)        ;CHECK FOR LAST PATTERN STORED SUCCESSFULLY
2781 176614 001031 bne    1900        ;br if last pattern is not correct
2782 176616 010210 MOV    R2,(R0)        ;STORE NEXT PATTERN

```

2783 176620 020210	OP	R2,(R0)	;CHECK FOR NEW PATTERN
2784 176622 001026	bn	1906	;br if new pattern NOT stored correctly
2785 176624 062700 000002	add	02,R0	;bump test address by 2
2786 176626 020005	OP	R0,R5	;CHECK FOR LAST ADDRESS REACHED
2787 176628 101767	BL06	306	;BRANCH IF MORE ADDRESSES TO TEST
2788 176630 005702	test	R2	when R2 = 177777 goto next step
2789 176632 100361	bnl	206	;br if more passes to make
2790			
2791			
2792			
2793			
2794			
2795			
2796			
2797			
2798			
2799			
2800			
2801			
2802			
2803			
2804 176640 010201	MOV	R2,R1	;SET R1 TO PATTERN TO CHECK FOR
2805 176642 005700	test	R0	;fastest way to clear the 'c' bit
2806 176644 006002	ror	R2	;add another zero to pattern in R2
2807 176646 010500	MOV	R5,R0	;SET R0 TO HIGHEST MEMORY ADDRESS
2808 176648 062700 000002	add	02,R0	;ADJUST ADDRESS FIRST TIME SO WE CAN USE DECREMENT INITIAL
LY 2809 176654 020100	700:	OP	R1,-(R0)
2810 176656 001010	bn	1906	;br if last pattern not stored correctly
2811 176658 010210	MOV	R2,(R0)	;STORE NEXT PATTERN (ONE ADDITIONAL 'ZERO-BIT')
2812 176662 020210	OP	R2,(R0)	;CHECK FOR PROPER STORAGE OF NEW PATTERN
2813 176664 001005	bn	1906	;br if new pattern NOT stored correctly
2814 176666 020004	OP	R0,R4	;CHECK FOR LOWEST MEMORY ADDRESS REACHED
2815 176670 101371	BNL	706	;BR IF STILL NOT AT LOWEST ADDRESS
2816 176672 005702	test	R2	when R2 = 0, the test is done
2817			
2818 176674 001361	clic		;side effect of clic instruction!
2819			
2820			
2821			
2822			
2823			
2824			
2825			
2826			
2827 176676 000207	rtb	PC	;exit with c bit clear
2828			
2829 176700	1906:	;(ERROR EXIT)	
2830 176700 000261	sec		
2831 176702 000207	rtb	PC	;return with c bit set

2033		.subttl Permanent Storage	
2034			
2035			; 'DIAGNOSE' command packet for TUEB
2036 176704	002	diagpk: .byte 2	; flag byte = 'command'
2037 176705	012	.byte 10.	; command byte count = ten
2038 176706	007	.byte diagpk	; op-code = 'diagnose'
2039 176707	000	.byte 0	; modifier byte = zero
2040 176710	000	.byte 0	; unit 0 = zero
2041 176711	000	.byte 0	; switches = zero
2042 176712	000	.byte 0	; seq 0 (low) = zero
2043 176713	000	.byte 0	; seq 0 (high) = zero
2044 176714	000	.byte 0	; data byte count (low) = zero
2045 176715	000	.byte 0	; data byte count (high) = zero

2047		;
2048		;
2049		;
2050		;
2051		;
2052		;
2053		;
2054		;
2055		;
2056		;
2057		;
2058		;
2059		;
2060		;
2061		;
2062		;
2063		;
2064		;
2065		;
2066		;
2067		;
2068	000062	remain=lasadd-.41 ;compute the 0 of bytes remaining or exceeded
2069		
2070		.if lt remain-2
2071		.error ;ROM size exceeded by remain bytes
2072		.iff
2073	176776	;-LAS00-1
2082	176776 000001	.word P0EDT+256.!POWER ;low byte = ROM version, High byte = ROM edit
2086		.end
2087		
2088	173000	.end coldst

ALLUP= 000037	HIPTST= 000040	K06TMR= 000054	RECUPK 176242	TU0070= 000006
ALTENT 173036	INILUP= 000020	K06TYP= 000043	REDAT 176052	TU50FL= 000023
ANVFAL 174372	INIT = 000004	K06URS= 000044	REMAIN= 000062	TU50RC 173030
BADHEM 174366	JMPTAR 173614	K06.PF= 000000	RESETS= 000005	TU50RD 173020
BADABO= 173000	JMPTR1 173630	K06.SZ= 000140	SAWERR 176544	TU50SN 173022
BLALUP= 000004	JMPTR2 173622	LADABO= 176777	SELP0 = 000400	TU50SY 173016
BLKSM = 002000	JSTST 174002	LCKBIT= 000001	SELP1 = 001000	T50FAL 175104
BTYPE2 173010	K00000= 000001	LEDBIT= 000010	SENDPK 176136	T50REA 175442
CKSWAC= 000416	K0000F= 000000	LDSIZ= 000010	SL0ER0= 000410	T50SST 175226
CKSWEX= 000414	K000UK= 001000	L0PTST= 000020	SL0ER1= 000412	T50SYC 175630
CLADIT= 000100	K000UP= 000400	LS = 063535	SL1ER0= 000400	T50TMO= 000110
CNOB = 000002	K0P0C1= 000001	MEMDON 174562	SL1ER1= 000402	T50TST 174664
CNTIE = 000002	K0P0P1= 000277	MEMEAR 174514	SLZER0= 000404	VECSIZ= 004000
COLDOT 173000	K0P0PS= 000076	MEMFAL= 000022	SLZER1= 000406	WRITE = 000003
CONT = 000020	K0P0SD= 000002	MEMPTY 174506	SNOWAT 176004	06LOBL= 000000
CPUFAL 174260	K0P0ST= 000203	MENTST 174354	S000AK 173566	0KPAR0= 172340
CRLST 175074	K0T0C1= 000001	MMENT 173024	S0P0PK= 002000	0KPAR4= ***** GX
DATA = 000001	K0T0W0= 000077	MLB/SM= 000522	S0P0RD= 004000	0KPAR5= ***** GX
DIADMB= 000007	K0T0P1= 000077	M000AT 173724	S0TTST 174372	0KPAR6= ***** GX
D10PK 176704	K0T0PS= 000076	M000T1 173730	TERMEN= 100000	0KPAR7= ***** GX
D0A0B= 000200	K0T0SD= 000002	M0VINV 176570	TESTND= 172340	0PMSR0= ***** GX
D0EP = 000100	K0T0ST= 000003	M0MBIT= 000004	TRMSFR 175112	0P10CS= ***** GX
D00RPT 173026	K0A0CU= 000377	PARSIZ= 040000	TUCKSM= 000005	0SL0RS= ***** GX
FLTUP= 000010	K0A0DU= 177400	PARTST 174436	TUERPT= 000422	0SL1RS= ***** GX
FLYSM = 004000	K0A0AB= 000000	P0WEDT= 000000	TUERR0= 000420	0SL2RS= ***** GX
F11ABC 174012	K0A0MB= 000052	P0WER= 000001	TUMER= 000421	0SMCSR= ***** GX
F11ABO 173636	K0A0CA= 000060	P10FAL= 000021	TU00M = 000001	0TUBRK= 000001
F1100M 174266	K0A0HP= 000056	P0S1TH= 000005	TUPROT= 000011	00END0= 000140
F11WTP 173734	K0A0DA= 000104	P1TIME0 174626	TURCER= 000010	00FLGS= 000000
F11TST 173032	K0A0FU= 000050	P1TIME1 174646	TURCTO= 000007	00PC00= 001001
H0LUP = 000001	K0A0FB= 000046	P1TIME2 175722	TUSLFT= 000003	00WALS= 000001
H0BM = 000400	K0A0LA= 000140	P1TIME3 176052	TUSOFT= 000004	00WALU= 177777
H0LUP = 000002	K0A0LT= 000042	READ = 000002	TUMER= 000420	..Y = 177400
H0BM = 001000	K00TLK= 000040	READUT 175362	TUSYMC= 000002	

.ABS. 177000 000
 000000 001

ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 11606 WORDS (46 PAGES)

DYNAMIC MEMORY: 19740 WORDS (75 PAGES)

ELAPSED TIME: 00:01:07

BAKEDJ:PIBOOT,BALST:PIBOOT/CL/-SP-BADHLE:K0CL1B/PL,B0UPAC:PIBOOT.JMC/EN:LC

For Internal Use Only

P10007 CREATED BY MACRO ON 16-MAY-82 AT 11:16 PAGE 1
SYMBOL CROSS REFERENCE CREF V01

SYMBOL	VALUE	REFERENCES
ALLUP	= 000037	010-439 27-1314 27-1327
ALERT	173036	12-524 014-615
ANAFAL	174572	12-543 22-1005 027-1290 31-1536
BADREN	174566	25-1219 26-1273 027-1281
BADREN	= 173000	07-299 7-334
BLKUP	= 000004	010-436 10-439 27-1303
BLKSM	= 002000	010-431
BTYP2	173010	012-523
CKSPAC	= 000416	07-328 7-329 049-2544
CKSPEX	= 000414	07-327 7-328 049-2545
CLABIT	= 000100	09-406
CPAD	= 000002	00-300 30-1913
CHTIE	= 000002	09-411
COLBOT	173000	012-503 57-2000
CONT	= 000020	00-302 00-2051
CPUFAL	174260	20-989 20-990 20-992 20-995 20-996 20-999 20-1001 20-1004 20-1006 21-1022 21-1033 21-1035 21-1037 21-1039 21-1042 22-1056 22-1057 22-1059 22-1060 22-1062 22-1063 22-1065 22-1066 022-1070 24-1165
CORLST	173074	30-1435 031-1535
DATA	= 000001	00-301 30-1963 47-2430
DIAGNO	= 000007	00-307 56-2030
DIOPAK	176704	35-1766 056-2036
EDUARD	= 000200	09-405
ENOP	= 000100	00-300 30-1965
ERRRPT	173026	012-543
FLTUP	= 000010	010-435 10-439 27-1303 27-1304
FLTSM	= 004000	010-430 27-1305
F11ABC	174012	19-964 020-971
F11ABO	173636	17-076 010-001
F11ODM	174266	22-1067 022-1007
F11HTP	173734	10-912 019-920
F11TST	173032	12-512 014-606
HALUP	= 000001	010-430 10-439 27-1303
HASH	= 000400	010-433
HOLUP	= 000002	010-437 10-439 27-1303
HDSH	= 001000	010-432
HIPTST	= 000040	09-407
INILUP	= 000020	010-434 10-439 22-1095
INIT	= 000004	00-379 30-2021
JPTAR	173614	17-072 017-074
JPTR1	173630	17-074 017-078
JPTR2	173622	017-076 17-078
JRTST	174002	19-962 019-967
LAGADO	= 176777	07-300 57-2068 57-2073
LXABIT	= 000001	09-412
LEDBIT	= 000010	09-409
LOBSIZ	= 000010	00-336 33-1652 36-1047
LOPTST	= 000020	09-408
MEMODM	174562	26-1247 026-1276
MEMERR	174514	26-1240 26-1242 026-1263
MEMFAL	= 000022	00-367 27-1209
MEMRTY	174506	026-1246 26-1274

For Internal Use Only

PRINTED BY MICRO ON 16-MAY-82 AT 11:16 PAGE 2
SYMBOL CROSS REFERENCE CREF U81

SYMBOL	VALUE	REFERENCES
ADJUST	174254	025-1176 25-1216 25-1222
ADJUST	173024	012-542
ALARM	000022	02-09
ALARM	173724	10-090 10-097 10-098 10-095 010-913 10-915 10-916
ALARM	173730	010-917 10-918
ALARM	176370	12-542 25-1191 26-1246 025-2754
ALARM	000044	09-410
ALARM	000000	00-354 26-1245 26-1271 29-1423 20-1779 26-1043
ALARM	174436	25-1192 026-1225
ALARM	000000	02-27 57-2002
ALARM	000001	02-27 57-2002
ALARM	000021	00-366 22-1004
ALARM	000005	00-306
ALARM	174626	027-1316
ALARM	174646	027-1320
ALARM	175722	040-2030
ALARM	176052	042-2149
ALARM	000002	00-304 20-1915
ALARM	175362	29-1430 30-1305 036-1034
ALARM	176242	12-544 30-1701 30-1951 046-2371
ALARM	176052	042-2140 47-2410 47-2422 00-2462 00-2474 49-2514 49-2526 50-2574 50-2506
ALARM	000062	057-2060 57-2070
ALARM	000005	09-440 14-015
ALARM	176544	30-1409 30-1507 053-2676
ALARM	000000	2-27 14-620 57-2075
ALARM	000400	09-404
ALARM	001000	09-403
ALARM	176136	12-541 35-1767 30-1940 043-2220
ALARM	000410	07-323 7-324
ALARM	000412	07-324 7-327
ALARM	000400	07-319 7-320 30-1436
ALARM	000402	07-320 7-321
ALARM	000404	07-321 7-322
ALARM	000406	07-322 7-323
ALARM	176004	30-2014 30-2016 30-2022 30-2024 041-2005 44-2264 44-2277 45-2300 45-2313
ALARM	173566	017-064 17-066
ALARM	002000	09-402 24-1161 24-1164 24-1160 24-1170 24-1171 25-1213 25-1215 25-1220
ALARM	004000	09-401 24-1161 24-1162 24-1164 24-1167 24-1160 24-1171 25-1210 25-1221
ALARM	174372	025-1193
ALARM	100000	09-400
ALARM	172340	07-302 014-616 024-1159 025-1175 026-1230 029-1412 033-1614
ALARM	175112	29-1451 31-1531 033-1605
ALARM	000005	09-426 49-2543
ALARM	000422	07-332 030-1456 030-1472 53-2677
ALARM	000420	07-329 7-330 7-331 7-332 030-1471 030-1504 53-2601
ALARM	000421	07-331 035-1706 030-1973
ALARM	000001	09-422 35-1000
ALARM	000011	09-430 30-1967
ALARM	000010	09-429 00-2040 42-2171
ALARM	000007	09-428 42-2157
ALARM	000003	09-424 35-1707
ALARM	000004	09-425 37-1064

For Internal Use Only

PIDOOT CREATED BY MACRO ON 16-NOV-82 AT 11:16

PAGE 3

SYMBOL CROSS REFERENCE

DEF U01

SYMBOL	VALUE	REFERENCES
TUNER	= 000420	07-330 035-1787 035-1800 037-1064 038-1967 040-2040 040-2053 041-2100 042-2157
TUSYNE	= 000002	042-2171 049-2543
TUSYTO	= 000006	09-423 40-2053
TUSOFL	= 000023	09-427 41-2100
TUSORC	173030	09-368 31-1535
TUSORD	173020	012-544
TUSORH	173022	012-540
TUSOSY	173016	012-541
TUSFAL	175104	012-539
TUSREA	175442	30-1474 031-1542
TUSGST	175226	12-540 36-1049 030-1056
TUSGYC	175630	29-1440 30-1487 035-1727
TUSYTH	= 000110	12-539 31-1520 35-1752 039-2000
TUSYST	174664	00-353 42-2149
VECSIZ	= 004000	26-1277 029-1412
WRITE	= 000003	00-355 25-1109 26-1239 26-1244 26-1245 26-1263
SKPARC	= 172300	00-305
SKPAR4	= 000000	07-301 7-302
SKPAR5	= 000000	021-1023 021-1031 21-1030 21-1041
SKPAR6	= 000000	19-960 20-903 22-1052 25-1211 29-1436 033-1615 033-1616 34-1667 34-1694
SKPAR7	= 000000	29-1430 29-1443 033-1617 34-1660
SKPAR8	= 000000	29-1440
SPICCS	= 000000	019-950
SQLORS	= 000000	24-1160 25-1212
SQLIRS	= 000000	31-1537
SQLIRS	= 000000	31-1535
SQLIRS	= 000000	31-1536
SQLIRS	= 000000	022-1095 27-1302
SQLIRS	= 000001	00-309 30-2012

100

0067 001

REFERENCES

ABOVE	02-69	
CHANCE	02-69	
CRASH	02-69	
CUPW	02-69	
DUPP	02-69	
FAULT	02-69	
MINIT	02-68	2-69
IDENT	02-2	2-27
IDENT1	02-12	2-27
KRM	02-68	2-70
MLB/BN	02-69	2-69
NETVET	02-69	
POLL	02-69	
POP	02-69	
PRECT	02-69	
PUSH	02-69	
RESET	02-69	
SMALL	02-69	
SMR20	02-69	
SMR30	02-69	
SMR40	02-69	
SMR50	02-69	
SPL	02-69	
SMALL	02-70	
.ABN.	02-69	
.BLK2	02-69	
.BLK2	02-69	
.FROM.	02-69	
.LOCK.	02-69	
.REG.	02-69	
.TO.	02-69	

T E F

CHAPTER 4

CHAPTER 4 HSC70 BOOT PROM LISTINGS

This chapter contains the Boot PROM listings for the HSC70. The page numbers for this chapter are in the upper right hand corner of the listings.

2-	1	===== HBC70 ROM Bootstrap version X83-01 =====
4-	1	Useful Information
5-	1	Useful Information (continued)
7-	1	Address Definitions
8-	1	Bit Definitions
11-	1	Alternate Entry Points and Pointer Table
13-	1	..TEST 0 - J-11 ISP Test
24-	1	MEMORY TESTS
25-	1	..TEST 1 - Program Memory (Swap Bits)
26-	1	..TEST 2 - Program Memory (Vector area)
27-	1	..TEST 3 - Program Memory (Partition area)
28-	1	FAULT Lamp Routine
29-	1	..TEST 4 - Rx33 Controller Hardware Test(s)
30-	1	..TEST 4 PART II Interrupts & DMA
33-	1	Register & hardware test error traps
34-	1	..TEST 6 - Find a good Rx33 drive.
37-	1	..TEST 7 - Transfer Control to Loaded Image
40-	1	RDTEST- perform a REDAL/VERIFY operation
42-	1	Read Boot Blocks to Memory
44-	1	RDREAD- read disk blocks to memory
47-	1	Rx33 Jack of All trades Subroutine
50-	1	Rx33 Local Interrupt Handler
51-	1	WAIT 33.2us Subroutine
52-	1	EXECOM - Common command interlock & execution routine
53-	1	Moving Inversions Test Description
54-	1	Moving Inversions Test

1 000000

```

ident 03,01,( ROM Bootstrap )
.title HBC70 ROM Bootstrap X83-01
.subtitle ***** HBC70 ROM Bootstrap version X83-01 *****
.ident /X8301/

```

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

```

+-----+
| JJJJJJJJJJJJ  11111  11111  RRRRRRRRR  XX  XX |
|      JJJJ      111  111  RR  RR  XX  XX |
|      JJJJ      111  111  RR  RR  XX  XX |
|      JJJJ      1111  111  RR  RR  XXXX |
|      JJJJ      1111  111  RRRRRRR  XX  XX |
|      JJJJ      111  111  RR  RR  XX  XX |
| JJ  JJJJ      1111  1111  RR  RR  XXX  XX |
| JJJJJJJJJJ  111111  111111  RR  RR  XXX  XX |
+-----+

```

```

J11 ROM BOOTSTRAP
(Rx33 FLOPPY DRIVE BASED)

```

```

;program name: HBC70 J11 ROM Bootstrap

```

```

;program abstract:

```

```

This program is designed to execute from the bootstrap ROM located
on the J-11 P.ioc module. The bootstrap consists of tests of the J-11
CPU, part of the Program P.ioc memory, and the Rx33, followed by
loading of the next part of the bootstrap sequence from the Rx33.
The purpose of the bootstrap tests is to provide confidence that
the P.ioc can be used to load further software via the Rx33.
The software loaded as the last action of the bootstrap program
consists of more extensive tests of the P.ioc module. After
these tests finish execution, the HBC70 Functional Software, or
the offline software is loaded. Program control then transfers to
one of these packages.

```

```

;copyright statement:

```

```

Copyright 1985, Digital Equipment Corporation

```

```

;disclaimer:

```

```

The information in this document is subject to change without
notice and should not be construed as a commitment by Digital
Equipment Corporation. Digital Equipment Corporation assumes
no responsibility for any errors that may appear in this doc-
ument.

```

```

The software described in this document is furnished under a
license and may only be used or copied in accordance with the
terms of such license.

```

```

Digital Equipment Corporation assumes no responsibility for the
use of reliability of its software on equipment that is not sup-
plied by Digital.

```

For Internal Use Only

MAC70 ROM Bootstrap X83-01 MACRO V05.03b Wednesday 06-Nov-85 11:12 Page 3

MAC70 ROM Bootstrap version X83-01

1
2
3
4
5
6
7
8
9
10
11
12
13

AUTHOR - VERSION - DATE LIST

Mike Hare	X81-00	Aug 1983	Original version with Te58 lead.
Red Lilah	X82-00	Feb 1985	Rx32 version.
Red Lilah	X82-01	Apr 1985	Rx33 version using DMA to load.
Red Lilah	X82-02	Apr 1985	Add EXJDAT, terminate command on error code.
Red Lilah	X82-03	Apr 1985	Added interrupt based read; this version only.
Red Lilah	X83-00	Oct 1985	Rewrite/optimize. DMA & interrupt tests added.
Red Lilah	X83-01	Nov 1985	Rev'd common execution timeout for rev. C J11.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

.tbl1 Useful Information

- 1) MAXIMUM PROGRAM SIZE - This program must fit into a 2,048 byte ROM.
(Locations 17773808 thru 17776777)
- 2) Special Assembly Files Required - None. This is a stand-alone program.
- 3) Special use of ROM PABs (page address registers) - ROM PABs are used as a convenient place to store information that must be passed along to the Init P.ioc test which succeeds the boot and are used as test locations. The contents of the USER PAB set are preserved. This is where boot reasons and failing addresses are stored.

PAB usage summary:

	USER	KERNEL
0	preserved	TEXT 0
1	preserved	initial stack
2	preserved	initial stack
3	preserved	initial stack
4	preserved	initial stack/register bit tests
5	preserved	unit 0 & type of boot (R3 on entry)
6	preserved	CSR of R23 to use (R4 on entry)
7	preserved	Validation of unit/type/CSR

- 4) Warm Boot entry restrictions: When using the 'WBOOT' boot feature, the ROM must be entered with interrupts disabled (PRI=7), with the ROM off, or in kernel mode with the I/O page mapped.
- 5) Processor-time-dependent loops - All instruction sequences labeled 'PTIMEs' are timing loops that depend on the basic cycle time of the J-11 CPU chip.
- 6) ROM VERSION NUMBER - The Version and Edit numbers of the ROM software are stored in the last word of the ROM (address 176776). The low byte is the version number, the high byte is the edit number.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

```
.sbit1 Useful information (continued)
```

```
7) EX33 ERROR CODE TABLE - This program constructs an error table which
remembers why the R33 subsystem could not be used for booting.
```

ADDRESS	MEANING
400	Contains controller error code (codes 1 & 2).
402	R33 address being accessed, if applicable.
404	Expected result.
406	Actual result.
410	Drive error code, byte-encoded : DRIVE_1/DRIVE_0

```
400 CONTROLLER ERROR| FAILURE INFORMATION
```

1	General. ROM occurred while accessing R33 registers.
2	A bit was stuck in the registers. See expected/actual.
3	Force mode interrupt did not occur.
4	BPA test mode hardware error.
5	BPA address counters wrong after transfer.
6	Incorrect data after BPA test operation.
7	Data parity bad after BPA test operation.

```
410 DRIVE ERRORS | FAILURE INFORMATION
```

10	Device not ready- (no disk, or door open).
11	Hard error, CRC -or- RBF on recal/verify.
12	Track_0 bit not set after recal.
13	Seek command timeout.
14	Seek error (CRC -or- RBF)
15	Read sector command timeout.
16	Hard error (CRC -or- RBF) on read.
17	Non-bootable image.

```
| In location 410 it is possible to have failure information for both drives.
| If this is the case you will have non-zero data in both bytes. Only in the case
| where failures are detected on both drives does the boot rom generate a
| 'LOADFAL' failure code and branch to the fault light routine.
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56

process: This test is initiated each time the P.10c subsystem is booted or rebooted. Different entry points to the ROM allow the software to determine whether the subsystem was being booted or rebooted. Testing begins with the ISP of the J-11 processor. All basic PDP-11 instructions and all eight addressing modes are tested. Following the J-11 ISP tests, 9 words of the P.10c Program memory are tested for proper addressing and ability to hold data and proper parity. The final test locates a working R23 drive that contains an HBC70 formatted floppy and reads the first 8 blocks from the diskette to the 8 word partition in the Program memory.

outputs:(to 'Init P.10c Test')

- case 1 - J-11 test passes, 8 word partition ok, R23 ok
R0 = unit number of R23 used for boot (lower byte)
R1 = type of boot (upper bit of upper byte) 0=cold, 1=warm
R2 = CSR address of R23 used for boot
R3 = Base address of tested 8 word partition
'INIT' lamp is on
- case 2 - J-11 test fails
No parameters passed. J-11 'hangs' immediately (branch self)
'Init' lamp does not light, 'fault' lamp off
- case 3 - No working 8 word partition found in Program Memory
(or first 2048 bytes of Program Memory are faulty)
'Init' lamp is lighted
'Fault' lamp is lighted
Depressing fault switch causes blinking fault code display
'Fault code' = (memory test failure)
HBC70 must be rebooted to attempt recovery
Program Memory/R23 controller module probably needs replacing
- case 4 - A failure was detected in the R23 controller logic test
'INIT' lamp is lighted
'Fault' lamp is lighted
Depressing fault switch causes blinking fault code display
'Fault code' = (memory test failure)
HBC70 must be rebooted to attempt recovery
Program Memory/R23 controller module probably needs replacing
... (error code table contains reason why each R23 unit failed. See 'R23 Error Table' under USEFUL INFORMATION (above))
- case 5 - No working R23 found with bootable media mounted
'Init' lamp is lighted
'Fault' lamp is lighted
Depressing fault switch causes blinking fault code display
'Fault code' = (R23 test failure)
HBC70 must be rebooted to attempt recovery
No R23 has bootable image mounted, or R23s are faulty
(R23 error code table contains reason why each R23 drive unit failed. See 'R23 Error Table' under USEFUL INFORMATION (above))

Address Definitions

```

1
2
3
4
5
6
7      172340      00P00= 172340      ; address of kernel P0R 0
8      172342      00P01= 172342      ; kernel P0R 1
9      172344      00P02= 172344      ; kernel P0R 2
10     172346      00P03= 172346      ; kernel P0R 3
11     172350      00P04= 172350      ; kernel P0R 4
12     172352      00P05= 172352      ; kernel P0R 5
13     172354      00P06= 172354      ; kernel P0R 6
14     172356      00P07= 172356      ; kernel P0R 7
15
16     177572      00000= 177572      ; MMU STATUS REGISTER 0
17     177574      00001= 177574      ; MMU STATUS REGISTER 1
18     177576      00002= 177576      ; MMU STATUS REGISTER 2
19
20     170040      0P10C= 170040      ; P.IOC CONTROL AND STATUS REGISTER
21     170042      00ACR= 170042      ; SWITCH REGISTER CSR
22
23
24
25
26
27     000400      EXTAB= 400          ; Unique code for controller
28     000410      EXTAB= 410          ; base of byte location for drive error table
29     172000      base= 172000        ; first byte in ROM
30     176777      last= 00000+3777    ; last byte in ROM
31     172340      TEST0= 00P00        ; kernel P0R 0 is used to hold the test number
32     000230      RXVEC= 230          ; R233 interrupt vector
33     000232      R2PSI = 232          ; R233 PSI on interrupt routine
34     000414      INTFL= 414          ; This will be the interrupt flag location
35

```

```

1      .sect1 Bit Definitions
2      ;-----
3      ;
4      ; Bit definitions
5      ;
6      ;-----
7
8      000001      bit0 = 1
9      000002      bit1 = 2
10     000004      bit2 = 4
11     000010      bit3 = 10
12     000020      bit4 = 20
13     000040      bit5 = 40
14     000100      bit6 = 100
15     000200      bit7 = 200
16     000400      bit8 = 400
17     001000      bit9 = 1000
18     002000      bit10 = 2000
19     004000      bit11 = 4000
20     010000      bit12 = 10000
21     020000      bit13 = 20000
22     040000      bit14 = 40000
23     100000      bit15 = 100000
24
25     040000      PMSIZ= 16384. ;size of program memory partition to be tested in bytes
26     004000      UESIZ= 2048. ;size of vector area to be tested in bytes
27     000010      LBSIZ= 0. ;# of blocks to load for boot
28
29     ;
30     ;-----
31     ;
32     ; Fault register code definitions
33     ;
34     ;-----
35     ;
36
37     000021      piofal=21 ; P.ioc module failure
38     000022      memfal=22 ; Memory module failure
39     000023      loadfl=23 ; load device/subsystem failure code
40
41     ;
42     ;-----
43     ;
44     ; P.ioc Control and Status Register Bit Definitions
45     ;
46     ;-----
47     ;
48
49     100000      TERMEN=100000 ;1 when terminal enable, 0 when Secure
50     004000      SHPCEN=004000 ;1 when Swap Control memory banks
51     002000      SHPPEN=002000 ;1 when Swap Private memory banks
52     001000      SELP1 =001000 ;Bit 1 of P.ioc interrupt code
53     000400      SELP0 =000400 ;Bit 0 of P.ioc interrupt code
54     000200      DMMARB=000200 ;1 when enable dmm arbitration
55     000100      CLABIT=000100 ;1 when clear I/O page device registers
56     000040      HPTST=000040 ;1 when enable high byte parity test
57     000020      LPTST=000020 ;1 when enable low byte parity test

```

For Internal Use Only

MBC70 ROM Bootstrap X83-01 MBC70 V03.03b Wednesday 06-Nov-85 11:12 Page 8-1

Bit Definitions

50	000010	LEDBIT=000010	;1 when light STATE LED
51	000004	IPMBIT=000004	;1 when create a non-memory-access cycle
60	000002	INTIE =000002	;1 when control memory interrupts enabled
61	000001	LCKBIT=000001	;1 when create a control memory lock cycle
62			
63			
64			
65			
66			
67			
68			
69			
70			
71	000005	RESETS= 000005	;define our own mnemonic for a RESET instruction
72			

Special local definition of POP-11 RESET instruction
(Required since RESET is illegal in normal MBC70 code,
and MBC70 Macro Library defines RESET to cause an error)

Bit Definitions

```

1
2
3
4
5
6
7
8
9
10      177400      RXCSR = 177400      ; RX32 CSR EQUATE
11      177402      RXTRK = 177402      ; CURRENT TRACK/LO HWR ADDRESS
12      177404      RXSEC = 177404      ; DESIRED SECTOR/NEXT HWR BYTE
13      177406      RXHWR = 177406      ; DESIRED TRACK/HIGH HWR ADDRESS
14      000017      MAXSEC = 15.
15      000117      MAXCYL = 79.
16      004302      MAXLEN = (MAXCYL*MAXSEC*2)
17      000017      PERsid = maxsec
18      000036      PERcyl = (PERsid*2)
19      ; command definitions sorted by type
20      001000      MTRDM = 1000      ; MOTOR ENABLE IS BIT9
21      021000      BDM = 21000      ; MOTOR ENABLE + INTERRUPT ENABLE
22      000000      DRV0 = 0      ; DRIVE SELECT_0 IS , NECESSARILY, 0
23      000400      DRV1 = 400      ; BUT DRIVE SELECT 1 IS NOT
24      010000      DISRQ = 10000      ; DISABLE REQUEST (NO DMA)
25      000004      VERIFY = 4      ; VERIFY AFTER STEP/SEEK OPERATION (CHECKS HEADER)
26      000020      MULTI = 20      ; MULTIPLE RECORD READ
27      002000      LED = 2000      ; TURN ON THE 'LED' ON THE CONTROLLER BOARD
28      040000      DMATST = 40000      ; DMA test mode bit
29      100000      MTRCK = 100000      ; Bit 15 to turn on the module on led in RDMR
30
31      ;positioning commands
32      000010      RECAL = 10      ; RESTORE TO TRACK 0
33      000020      SEEK = 20      ; SEEK WITH HEADS UNLOADED AT START
34      000030      SEEKL = 30      ; seek with heads loaded at start
35      000060      STEP = 60      ; STEP WITH TRACK UPDATE (JUST UPDATES COUNT UNLESS VERIFY SET)
36      000120      STEPIN = 120      ; STEP IN WITH TRACK UPDATE. DITTO THE ABOVE
37      000160      STEPOUT = 160      ; STEP OUT WITH TRACK UPDATE. DITTO THE DITTO ABOVE
38
39      ;interrupt mode commands.
40      000320      RSINT = 320      ; reset interrupt modes
41      000330      FRCINT = 330      ; FORCE UNCONDITIONAL INTERRUPT
42      000324      INTON = 324      ; INTERRUPT ON EACH INDEX PULSE
43      000323      INTDY = 323      ; INTERRUPT ON DRIVE COMING READY AFTER OPERATION (not ready)
44
45      ;data transfer commands.
46      000000      SIOE0 = 0      ; VALUES TO UPDATE SIDE SELECT
47      000002      SIOE1 = 2
48      000214      RDSEC = 214      ; READ SECTOR BASIC COMMAND ;REQUIRES MORE ARGUMENTS
49      000254      WRSEC = 254      ; WRITE SECTOR BASIC COMMAND ; ALSO REQUIRES EXTRA ARGUMENTS
50      000304      RDHWR = 304      ; READ ADDRESS (HEADER)
51      000340      RDTRK = 340      ; READ TRACK
52      000360      WRTRK = 360      ; WRITE TRACK
53
54      ; Frequently used command combinations.
55      001020      FDD = MTRDM+SEEK+VERIFY      ; what you need to seek correctly. A LOCAL
56      021020      SHV = BDM+SEEK+VERIFY      ; SEEK+VERIFY+ MOTOR_ENABLE+INTERRUPT_ENABLE
57

```

For Internal Use Only

MSC70 RMI Bootstrap X83-01 MICRO V85.03b Wednesday 05-Mar-85 11:12 Page 9-1

Bit Definitions

30		; Status values.	
30	100000	RDPE = 100000	; memory parity error
31	000000	RD001 = 000000	; memory 1001
32	000010	CRCEA = 10	; crc error on data transfer
33	000020	RDPER = 20	; record not found
34	000020	SEEKER = 20	; seek error for positioning commands
35	000200	RD00Y = 200	; NOT READY when set
36	000004	RD0LT = 4	; data late
37	000001	RD0SY = 1	; busy bit
38	000004	TRACK0 = 4	; track 0 status bit for seek/recal commands
		----->	

1			1
2			1
3			1
4			1
5			1
6			1
7			1
8			1
9	004000	FLTSW	=004000 ;1 when 'Fault' switch depressed
10	002000	OLSW	=002000 ;1 when 'Online' switch depressed
11	001000	F1SW	=001000 ;1 when 'First Unmarked' switch depressed
12	000400	F2SW	=000400 ;1 when 'Second Unmarked' switch depressed
13	000020	INIUP	=000020 ;1 when light 'Init' lamp
14	000010	FLTUP	=000010 ;1 when light 'Fault' lamp
15	000004	OLUP	=000004 ;1 when light 'Online' lamp
16	000002	F1UP	=000002 ;1 when light 'First Unmarked' lamp
17	000001	F2UP	=000001 ;1 when light 'Second Unmarked' lamp
18	000037	ALLUP	=F1UP F2UP OLUP FLTUP INIUP ;inclusive OR of all lamp bits
19			
20			

Alternate Entry Points and Pointer Table

```
1      .sect1 Alternate Entry Points and Pointer Table
2
3      ;+
4      ;-----
5      ;
6      ; Alternate entry points to this program are supplied to allow
7      ; the distinction between a 'cold' boot and a 'warm' boot.
8      ; (A cold boot is the result of power-up, or '1730000' to ODT. No
9      ; saved context exists for a cold boot. A warm boot indicates that
10     ; there is saved context in the User PAR set, to be used by the last
11     ; P.ioc test.) The forced boot mechanism is preserved from the old boot
12     ; ROM for compatibility.
13     ;
14     ;inputs:
15     ;      R3 = unit # of R33 to use for boot (lower byte)
16     ;      R4 = CSR address of R33 to use for boot (16 bits)
17     ;      R5 = Validation of unit/CSR (R3 XOR R4)
18     ;      For Warm boots, the User PAR set contains the reason
19     ;      for the reboot, and other context. This context is not
20     ;      used by the Bootstrap ROM, it is merely not disturbed.
21     ;
22     ;process:
23     ;      Set up a 'type of boot' parameter, depending upon which
24     ;      entry into the ROM is used. The following entry points
25     ;      are provided:
26     ;          1) Entry for 'Cold' boot
27     ;          2) Entry for 'Warm' boot
28     ;
29     ;outputs:
30     ;      R3, R4, and R5 preserved
31     ;      'Type of boot' encoded in upper bit of R3
32     ;
33     ;      The value of the upper bit of R3 is as follows:
34     ;          cold boot = 0
35     ;          warm boot = 1
36     ;
37     ;notes:
38     ;
39     ;History of changes to this routine
40     ;-----
41     ;-
```

Alternate Entry Points and Pointer Table

```

1 000000      .sect
2      173000      .bssadd
3
4      |-----|
5      |
6      |      Entry for 'Cold' boot
7      |      This is starting address as defined by the hardware after power-up or INIT
8      |-----|
9      |~
10
11 173000      coldboot:
12
13 173000 106427 000340      mtps 0340      ;prevents real time clock from interrupting
14 173004 000167 000022      jmp  J11TST      ;cold boot entry
15
16
17      |-----|
18      |
19      |      Entry for 'Warm' boot      (base address + 10)
20      |-----|
21      |~
22
23
24
25 173010 052703 100000      BTYPE2: bsr  0100000,R3      ;set R3 negative to indicate warm boot
26 173014 000410      br      ALTDEF      ;start bootstrap tests
27
28
29      |-----|
30      |
31      |      Table of pointers to ROM routines that are used from ROM
32      |      (by the INIT P.LOC test).
33      |
34      |      The relative position of these pointers, with respect to the
35      |      starting address of the ROM, must NEVER be changed.
36      |-----|
37      |~
38
39
40
41 173016 175722      Rx33RY: .word  RXREST      ;base + 16      Rx33 restore/synchronization routine
42 173020 176132      Rx33RD: .word  RXREAD      ;base + 20      Rx33 read routine
43 173022 176770      Rx33SN: .word  SENDPK      ;base + 22      Send Rx33 command packet (dummy vector now)
44 173024 176636      NIMENT: .word  NID/INI      ;base + 24      Moving Inversions ROM test
45 173026 174634      ERRAPT: .word  ANYTAL      ;base + 26      Fault code display routine
46 173030 176770      Rx33RC: .word  RECUPK      ;base + 30      Receive Rx33 command or data packet (do now)
47
48      ;***** END OF POINTER TABLE *****
49

```

..TEST 0 - J-11 ISP Test

```

1      .title ..TEST 0 - J-11 ISP Test
2      ;*
3      ;*****
4      ; Test the basic POP-11 instruction set and addressing modes.
5      ;
6      ;inputs:      PSM Priority field = 7 (interrupts disabled) upon entry
7                   R3 = Unit 0 to use for boot (lower byte, value of 0 or 1 only)
8                   R4 = CSR address of R3 to use for boot (16 bit I/O page address)
9                   R5 = Validation for R3 and R4 (R3 XOR R4)
10                  ;
11                  ; (validation prevents spurious device selection)
12      ;
13      ;Process: Test the Instruction Set Processor (ISP) of the P.10c's J-11
14               ;processor.
15               ;Instructions tested:
16               ;CLR(B),CDR(B),INC(B),DEC(B),NEB(B),TST(B),ROR(B),ROL(B),
17               ;ADR(B),ABL(B),SHR,BOP,ADC(B),CCC,CLN,CLV,CLZ,SCC,SDN,SEV,SEZ
18               ;
19               ;MOV(B),OP(B),BIT(B),BIC(B),BIS(B),ADD,SUB
20               ;
21               ;BR,BNE,BEQ,BPL,BMI,BCC(BMI),BCS(BLO),BNE
22               ;BLT,BGT,BLE,BHI,BLOS,BVC,BVS
23               ;
24               ;JMP,JSR,RTS,SRH,HTPS,HFPS
25               ;
26               ;Addressing modes tested:
27               ;All eight addressing modes are tested
28               ;
29               ;Instructions NOT tested:
30               ;ASH,ASHC,BPT,EXT,HALT
31               ;IOT,MAK,HFPI,HTPI,MUL,RESET,RTI,RTT,SBC
32               ;BPL,TRAP,WAIT,XOR
33               ;
34               ;J-11 Logic NOT tested:
35               ;Memory Mapping, Interrupt priority and arbitration,
36               ;Stack Overflow, Power Fail, Bus Time-out Logic
37               ;
38               ;P.10c logic NOT tested
39               ;Control Memory Windows, ECC tables
40               ;Operator console interface, 'R' status registers
41               ;Control and arbitration of Data and Control buses
42               ;Remainder of Program Memory (total size minus 8 words)
43               ;
44      ;outputs:
45      ;Case 1 - no failure detected
46               ;'Init' lamp is lighted
47               ;Control passed to Program Memory Test
48               ;R3,R4, and R5 are preserved in P10 Kernel PAR's 5,6, and 7
49               ;
50      ;Case 2 - failure detected
51               ;J-11 CPU is 'hung'. PC of 'hung' identifies failure
52               ;'Init' lamp is out, 'Fault' lamp is out
53      ;
54      ;history of changes to this test:
55      ; 21-Oct-85 added mini divide test as part of CPU test
56      ;*****
57      ;*

```


..TEST 0 - J-11 ISP Test

```

1
2
3
4
5
6
7 173032 042703 100000 J11TST: bic 0100000,R3 ;set 'type of boot' to 'cold boot'
8 173036 000003 ALTBIT: RESETS ;clear all OCP lamps, clear all R interrupts
9 173040 012737 000414 177746 MBV 0414,00177746 ;disable cache
10 173046 003047 177266 clv TEST00 ;set test 0 to 0
11 173052 003006 clv SP ;SP gets a 0
12 173054 003106 con SP ;SP gets 177777, causes DBL BUS ERR if trap occurs
13
14 ; MBV 032,00177564 ; & & & temporary set up for 9600 baud console
15
16
17
18
19
20
21
22
23
24
25
26
27
28 173056 000277 ; M (- 1, Z (- 1, V (- 1, C (- 1
29 173060 001377 jnb (no branch expected)
30 173062 100377 jnb
31 173064 103377 jnb
32 173066 003377 jnb
33 173070 101377 jnb
34 173072 002777 jnb
35 173074 102377 jnb
36 173076 000250 jn (- 0
37 173100 002377 jnb (n clear, v set)
38 173102 000270 jn (- 1
39 173104 002777 jnb (n set, v set)
40 173106 000241 jc (- 0
41 173110 103777 jnb
42 173112 000261 jc (- 1
43 173114 103377 jnb
44 173116 000244 jz (- 0
45 173120 001777 jnb
46 173122 000264 jz (- 1
47 173124 001377 jnb
48 173126 000242 jv (- 0
49 173130 102777 jnb
50 173132 000262 jv (- 1
51 173134 102377 jnb
52 173136 000257 ; M (- 0, Z (- 0, V (- 0, C (- 0
53 173140 001777 jnb (no branch expected)
54 173142 100777 jnb
55 173144 103777 jnb
56 173146 002777 jnb
57 173150 003777 jnb

```

50	173152	101777		bior	.		;nbe
50	173154	102777		bvs	.		;nbe
60	173156	000401		br	10		;branch forward
61	173160	000777		br	.		;failed to branch forward
62	173162	000270	10:	sen	.		;set the N bit (N ← 1)
63	173164	100377		bpl	.		;nbe (no branch expected)
64	173166	002377		bys	.		;nbe
65	173170	003377		bys	.		;nbe
66	173172	002402		bit	20		;should branch
67	173174	000777		br	.		;previous branch failed
68	173176	003402	30:	bis	100		;should branch
69	173200	100776	20:	bis	30		;should branch backwards
70	173202	000777		br	.		;backwards branch failed
71							
72							
73	173204		100:	{			
74							
75							
76							Test CLR, COM, INC, DEC, NEG, TST, ROR, ROL, ASR, ARL, SWAB, NOP
77							and SEC
78							
79							Address mode 0, using R0
80							
81							
82							
83							
84	173204	005000		clr	R0		; R0 ← 0, C ← 0
85	173206	005100		com	R0		; R0 ← 177777, C ← 1
86	173210	005200		inc	R0		; R0 ← 000000, C = 1 (C not affected by inc)
87	173212	005300		dec	R0		; R0 ← 177777, C = 1 (C not affected by dec)
88	173214	000240		nop			;ensure nop does not interfere
89	173216	103377		bcc	.		;hang here if c bit is clear, or bcc fails
90	173220	005400		neg	R0		; R0 ← 000001, C ← 1
91	173222	103377		bcc	.		;hang here if c bit is clear, or bcc fails
92	173224	005700		tst	R0		; R0 = 000001, C ← 0
93	173226	001777		beq	.		;hang here if R0 = 0, or tst or beq fails
94	173230	006000		ror	R0		; R0 ← 000000, C ← 1
95	173232	001377		bne	.		;hang here if R0 not 0
96	173234	103377		bcc	.		;hang here if C didn't get loaded with a 1
97	173236	006100		rol	R0		; R0 ← 000001, C ← 0
98	173240	000240		nop			;ensure nop does not interfere
99	173242	103777		bcs	.		;hang if c bit is set
100	173244	001777		beq	.		;hang if R0 is 0
101	173246	000261		sec			; R0 = 000001, c ← 1
102	173250	000300		swab	R0		; R0 ← 000400, C ← 0
103	173252	103777		bcs	.		;hang if swab did not clear c bit
104	173254	105700		tsb	R0		;test for lower byte of R0 = 0
105	173256	001377		bne	.		;hang here if tsb failed, or R0 (lower) not 0
106	173260	000300		swab	R0		; R0 ← 000001, C ← 0
107	173262	105700		tsb	R0		;test that lower byte of R0 now not zero
108	173264	001777		beq	.		;hang if tsb failed, or R0 (lower) = 0

1		+	
2		-----	
3		:	
4		:	Test MOV, CMP, BIT, BIC, BIS, ADD, SUB
5		:	
6		:	Address mode 0, using R0, R1, and SP (R6)
7		:	
8		-----	
9		-	
10			
11	173266 010000	mov	SP, R0 ; R0 (- 177777
12	173270 020000	cmp	SP, R0 ; R0 = 177777, SP = 177777
13	173272 001377	bne	. ;hang if SP and R0 not equal
14	173274 040000	bic	SP, R0 ; R0 (- 000000
15	173276 001377	bne	. ;hang if R0 not 0
16	173300 020000	cmp	SP, R0 ; R0 = 000000, SP = 177777
17	173302 001777	beq	. ;hang if compare failed
18	173304 005200	inc	R0 ; R0 (- 000001
19	173306 040000	add	SP, R0 ; R0 (- 000000, C (- 1
20	173310 001377	bne	. ;hang if add failed
21	173312 103377	bcc	. ;hang if c bit did set
22	173314 030000	bis	SP, R0 ; R0 (- 177777
23	173316 005100	com	R0 ; R0 (- 000000
24	173320 001377	bne	. ;hang if R0 not zero
25	173322 160000	sub	SP, R0 ; R0 (- 000001
26	173324 001777	beq	. ;hang if R0 = 0
27	173326 005300	dec	R0 ; R0 (- 000000
28	173330 001377	bne	. ;hang if R0 not 0
29	173332 005200	inc	R0 ; R0 (- 000001
30	173334 005001	clr	R1 ; R1 (- 000000
31	173336 005201	inc	R1 ; R1 (- 000001
32	173340 040001	bic	R0, R1 ; R1 (- 000000
33	173342 001377	bne	. ;hang if R1 not 0
34	173344 005201	inc	R1 ; R1 (- 000001
35	173346 005100	com	R0 ; R0 (- 177776
36	173350 060100	add	R1, R0 ; R0 (- 177777, C (- 0
37	173352 103777	bcs	. ;will hang if C nbit set
38	173354 005100	com	R0 ; R0 (- 000000
39	173356 001377	bne	. ;hang if R0 not 0

```

1
2
3
4
5
6
7
8
9
10
11
12 173300 005100      com    R0          ; R0 (- 177777
13 173362 110001      movb   R0,R1       ; R1 (- 177777 (movb to register sign extends)
14 173364 005101      com    R1          ; R1 (- 000000
15 173366 001377      bne     .          ;hang if R1 not 0
16 173370 110100      movb   R1,R0       ; R0 (- 000000 (movb to register sign extends)
17 173372 001377      bne     .          ;hang if R0 not 0
18 173374 105100      comb   R0          ; R0 (- 000377
19 173376 105700      tctb   R0          ; R0 = 000377
20 173400 001777      beq     .          ;hang if lower byte is zero
21 173402 000300      movb   R0          ; R0 (- 177400
22 173404 105700      tctb   R0          ; R0 = 177400 (lower byte = 0)
23 173406 001377      bne     .          ;hang if lower byte of R0 not 0
24 173410 005101      com    R1          ; R1 (- 177777
25 173412 105001      clrb   R1          ; R1 (- 177400
26 173414 105701      tctb   R1          ; R1 = 177400
27 173416 001377      bne     .          ;hang if lower byte of R1 not 0
28 173420 000301      movb   R1          ; R1 (- 000377
29 173422 105701      tctb   R1          ; R1 = 000377
30 173424 001777      beq     .          ;hang if lower byte of R1 = 0
31 173426 120001      cmpl   R0,R1       ; R0 = 177400, R1 = 000377
32 173430 001777      beq     .          ;hang if lower bytes are equal
33 173432 000300      movb   R0          ; R0 (- 000377
34 173434 106300      andb   R0          ; R0 (- 000376, C (- 1
35 173436 103377      bcc     .          ;hang if c bit is not set
36 173440 120001      cmpl   R0,R1       ; R0 (lower) = 376, R1 (lower) = 377
37 173442 001777      beq     .          ;hang if lower bytes are equal
38 173444 105200      ractb  R0          ; R0 (- 000377
39 173446 006300      orl     R0          ; R0 (- 000776
40 173450 105200      ractb  R0          ; R0 (- 000777
41 173452 120001      cmpl   R0,R1       ; R0 = 000777, R1 = 000377 (equal in lower byte)
42 173454 001377      bne     .          ;hang if lower bytes are not equal
43 173456 105300      decb   R0          ; R0 (- 000776
44 173460 106200      arcb   R0          ; R0 (- 000777 (lower byte sign extends)
45 173462 120001      cmpl   R0,R1       ; R0 = 000777, R1 = 000377 (equal in lower byte)
46 173464 001377      bne     .          ;hang if lower bytes are not equal
47 173466 100100      bicb   R1,R0       ; R0 (- 000400 (lower byte cleared)
48 173470 001377      bne     .          ;hang if lower byte did not result in a 0
49 173472 120100      bicb   R1,R0       ; R0 (- 000777
50 173474 105100      comb   R0          ; R0 (- 000400 (lower byte 0)
51 173476 001377      bne     .          ;hang if lower byte not 0
52 173500 120100      bitb   R1,R0       ; R0 = 000400, R1 = 000377
53 173502 001377      bne     .          ;hang if bit test result was a 1
54 173504 105200      ractb  R0          ; R0 (- 000401
55 173506 105101      comb   R1          ; R1 (- 000000
56 173510 105201      ractb  R1          ; R1 (- 000001
57 173512 120100      bitb   R1,R0       ; R0 = 000401, R1 = 000001

```

For Internal Use Only

MB270 ROM Bootstrap X03-01 MICRO V85.03b Wednesday 06-Nov-05 11:12 Page 16 - 1

..TEST 0 - J-11 ISP Test

30 173514 001777	beq	.	hang if bit test result was zero
31 173516 000241	clc	.	; C ← 0
32 173520 106001	rorb	R1	; R1 ← 000000, C ← 1
33 173522 103377	bcc	.	hang if c bit is not a 1
34 173524 001377	bne	.	hang if result in R1 not 0
35 173526 106101	rolb	R1	; R1 ← 000001, C ← 0
36 173530 103777	bcs	.	hang if c bit is set
37 173532 001777	beq	.	hang if result in R1 is 0
38 173534 105401	negb	R1	; R1 ← 000377, c ← 1
39 173536 105201	incb	R1	; R1 ← 000000, C remains 1 (not affected)
40 173540 103377	bcc	.	hang if c bit clear
41 173542 001377	bne	.	hang if result in R1 is not zero
42 173544 000300	sub	R0	; R0 ← 000401
43 173546 105300	decb	R0	; R0 ← 000400
44 173550 001377	bne	.	hang if result in R0 (lower) is not 0
45 173552 005700	tst	R0	; R0 = 000400
46 173554 001777	beq	.	hang if result in R0 is 0

```

1
2
3
4
5
6
7
8
9
10
11 173536 005000      clr      R0          ; R0 (- 000000)
12 173560 005001      clr      R1          ; R1 (- 000000)
13 173562 005002      clr      R2          ; R2 (- 000000)
14 173564 005200      inc      R0          ; R0 (- 000001)
15 173566 005201      inc      R1          ; R1 (- 000001)
16 173570 006300      asl      R0          ; R0 (- 000002)
17 173572 006301      asl      R1          ; R1 (- 000002)
18 173574 105301      subhsh: decb     R1          ; R1 (- R1 minus 1)
19 173576 105202      incb     R2          ; R2 (- R2 plus 1)
20 173600 077003      sub      R0,subhsh    ; R0 (- R0 minus 1, branch if result not 0)
21 173602 105701      rshb     R1          ; R1 should now be zero also
22 173604 001377      bne      .           ;hang if R1 is not 0
23 173606 105302      decb     R2          ; R2 (- 1)
24 173610 105302      decb     R2          ; R2 (- 0)
25 173612 001377      bne      .           ;hang if R2 did not equal 2 after loop above
26 173614 000167 000002      jmp      jmptr      ;test a forward jump
27 173620 000777          br      .           ;hang if jump failed
28 173622 000167 000010      jmptr: jmp      jmptr1    ;test another forward jump
29 173626 000777          br      .           ;hang if jump failed
30 173630 000167 000010      jmptr2: jmp      J11A00    ;test another forward jump
31 173634 000777          br      .           ;hang if jump failed
32 173636 000167 177766      jmptr1: jmp      jmptr2    ;test a backward jump
33 173642 000777          br      .           ;hang if jump failed

```


..TEST 0 - J-11 ISP Test

```

1
2
3
4
5
6
7
8
9
10
11
12
13 173644 012710 173732 J11A00: mov 0moddat,R0 ;R0 gets pointer to data for this test (912)
14 173650 011001 mov (R0),R1 ;R1 gets test data (911)
15 173652 022001 cmp (R0)+,R1 ;check data against itself (912)
16 173654 001377 bne . ;hang if datum not equal
17 173656 020027 173730 cmp R0,0MODDAT+2 ;check R0 for correct contents
18 173662 001377 bne . ;hang if R0 is wrong
19 173664 063001 add 0(R0)+,R1 ;add test data to R1 (913)
20 173666 020027 173736 cmp R0,0MODDAT+4 ;check R0 for correct value
21 173672 001377 bne . ;hang if wrong value in R0
22 173674 163001 sub 0-(R0),R1 ;subtract test data from R1 (915)
23 173676 040001 bic -(R0),R1 ;clear R1 by bit clear with same data (914)
24 173700 001377 bne . ;hang if R1 was not cleared
25 173702 020027 173732 cmp R0,0MODDAT ;check R0 for correct value
26 173706 001377 bne . ;hang if R0 is incorrect
27 173710 054001 000004 btl 4(R0),R1 ; R1 (- 177777) (916)
28 173714 037001 000006 bit 06(R0),R1 ;test for any bits set (917)
29 173720 001777 beq . ;hang if no bits are set
30 173722 005101 com R1 ; R1 (- 000000)
31 173724 001377 bne . ;hang if R1 was not all ones before com
32 173726 000405 br J11HTP ;transfer to next test
33 173730 000777 br . ;previous branch failed
34
35 173732 173732 moddat: .word moddat ;data locations for addressing mode test
36 173734 173732 .word moddat
37 173736 177777 modd1: .word 177777
38 173740 173736 .word modd1

```

..TEST 0 - J-11 IOP Test

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 173742 000277
19 173744 106700
20 173746 020027 177757
21 173752 001377
22 173754 000257
23 173756 106700
24 173760 020027 177740
25 173764 001377
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40 173766 005037 177572
41 173772 012706 172352
42
43 173776 004767 000006
44 174002 000777
45 174004 000485
46 174006 000777
47
48 174010 062716 000002
49 174014 000207
50 174016 000777

```

;+

 ;
 ; Test IOPS Instruction
 ;
 ; Address mode 0
 ;
 ; Note: Until interrupt logic is tested (in 'Init P.roc test'
 ; following the ROM test), testing of the priority bits
 ; in the PBI are delayed to avoid uncontrolled detection
 ; of failures in the interrupt logic. This test restricts
 ; itself to checking that the priority is now 7, and that
 ; the N, Z, V, and C bits are set and cleared.
 ;
 ;-----
 ;-

```

JLINTP: scc      ; N (- 1, Z (- 1, V (- 1, C (- 1
         mps     R0      ; R0 (- 177757 (lower byte of PBI, sign extend)
         cmp     R0,017757 ; check for pri 7, all cc bits set
         bne     .        ; hang if mps failed
         ccc     .        ; N (- 0, Z (- 0, V (- 0, C (- 0
         mps     R0      ; R0 (- 177740
         cmp     R0,017740 ; check for pri 7, all cc bits clear
         bne     .        ; hang if mps failed
         ;+
         -----
         ;
         ; Test JSR and RTS instructions
         ; TURN OFF THE PMU
         ;
         ; NOTE: In order to test JSR, the stack must be used. This
         ; requires using some location in memory for a one word stack.
         ;
         ; This test uses part of the PMU's Kernel Page Address Register Set as
         ; a stack area (8 contiguous 16 bit registers).
         ;-----
         ;-
         clr     00000000 ;TURN OFF THE PMU
         mov     00KPAR5,SP ;point SP to Kernel PARS
         ;(KPAR's 4,3,2,1, and 0 used for stack)
         jsr     PC,JSRTST ;test JSR instruction. Return at call+6
         br     .          ;hang if JSR returns to wrong location
         br     J11ADC     ;Correct return from JSR above
         br     .          ;hang if JSR returned to wrong location
         ;-----
         ;-
         JSRTST: add    02,(SP) ;update return address by 2
         rts     pc         ;test the RTS instruction
         br     .          ;hang if RTS failed to return

```

1
2
3
4
5
6
7
8
9
10
11
12

1+

1
1
1
1
1
1
1
1
1
1
1
1-

Test ABC and ABCB instructions

Move the test parameters from R3, R4, and R5 to the Memory Management Kernel PAR's 5, 6, and 7, where they will be preserved across the memory and Rx33 tests to follow.

13 174020
14 174020 012701 172352
15 174024 010321
16 174026 010421
17 174030 010311
18 174032 005001
19 174034 005301
20 174036 001130
21 174040 103527
22 174042 103501
23 174044 001125
24 174046 005101
25 174050 005301
26 174052 001122
27 174054 103121
28 174056 012702 177777
29 174062 105502
30 174064 103115
31 174066 020227 177400
32 174072 001112
33 174074 000261
34 174076 005301
35 174100 103507
36 174102 005301
37 174104 001105

J11ABC:

```

mov    00KPAR5,R1    ;point R1 to Kernel par 5
mov    R3,(R1)+      ;save unit and type of test in UPAR5
mov    R4,(R1)+      ;save CSR of Rx33 in UPAR6
mov    R5,(R1)       ;save validation word for unit and CSR
clr    R1            ; R1 = 0, C = 0
adc    R1            ; R1 = 0      (0+0=0)
bne    CPUFAL        ;br if answer is non-zero
bcs    CPUFAL        ;br if c bit is set
adcb   R1            ; R1 = 0      (0+0=0)
bne    CPUFAL        ;br if answer is non-zero
cm     R1            ; R1 = 177777, C = 1
adc    R1            ; R1 = 0, C = 1 (177777+1=0)
bne    CPUFAL        ;br if answer is not zero
bcc    CPUFAL        ;br if c bit did not set
mov    0177777,R2    ; R2 = 177777, C = 1
adcb   R2            ; R2 = 177400, C = 1
bcc    CPUFAL        ;br if c bit did not set
cmp    R2,0177400    ;check for correct answer in R2
bne    CPUFAL        ;br if wrong answer
sec    C             ; C = 1
adc    R1            ; R1 = 1, C = 0 (0+1=1)
bcs    CPUFAL        ;br if c bit is set
dec    R1            ; R1 = 0      (1-1=0)
bne    CPUFAL        ;br if R1 did not contain a 1

```

155

```

1
2
3
4
5
6
7
8
9 174212 012706 172352      mov     00KPA05,SP      ;set stack pointer to P00 Kernel PA05
10 174216 003001             clr     R1                  ; R1 = 0
11 174220 062701 125252      add     0125252,R1       ; R1 = 125252, N=1, Z=0, U=0, C=0
12 174224 062701 052525      add     0052525,R1       ; R1 = 177777, N=1, Z=0, U=0, C=0
13 174230 103433             bcs     CPUFAL          ;br if c bit is set
14 174232 003032             bgt     CPUFAL          ;br if a bit is not set
15 174234 102431             bvs     CPUFAL          ;br if v bit is set
16 174236 062701 146314      add     0146314,R1       ; R1 = 146313, N=1, Z=0, U=0, C=1
17 174242 103026             bcc     CPUFAL          ;br if c bit is not set
18 174244 062701 125252      add     0125252,R1       ; R1 = 073565, N=0, Z=0, U=1, C=1
19 174250 102023             bvc     CPUFAL          ;br if v bit did not set
20 174252 103022             bcc     CPUFAL          ;br if c bit did not set
21 174254 062701 104213      add     0-73565,R1       ; R1 = 000000, N=0, Z=1, U=0, C=1
22 174260 001017             bne     CPUFAL          ;br if R1 not zero
23 174262 102416             bvs     CPUFAL          ;br if v bit is set
24 174264 103015             bcc     CPUFAL          ;br if c bit is clear, else do divide test
25
26
27
28
29
30
31 174266 012702 177777      DIVTEST: MOV     0-1,R2
32 174272 010203             MOV     R2,R3                  ; R2, R3 both have minus one
33 174274 071227 177777      DIV     0-1,R2                  ; perform the divide
34
35 174300 100407             BHI     CPUFAL          ; check the flag state now
36 174302 103406             BCS     CPUFAL
37 174304 102405             BVS     CPUFAL
38
39 174306 005703             TST     R3                  ; should be no remainder
40 174310 001003             BNE     CPUFAL
41
42 174312 020227 000001      CMP     R2,01                  ; resulting quotient
43 174316 001403             BEQ     J1100M          ; if not what we got

```

For Internal Use Only

NBC70 ROM Bootstrap X83-01 MACRO V05.03b Wednesday 06-Nov-83 11:12 Page 23

..TEXT 0 - J-11 ISP Test

```
1      ;  
2      ;-----  
3      ;  
4      ; J-11 CPU test failure (only used for last few J-11 tests)  
5      ;  
6      ; 1) Set to 00 to fault code for "P.roc failure"  
7      ;  
8      ; 2) light the FAULT lamp  
9      ;  
10     ; 3) Monitor the FAULT switch  
11     ;-----  
12     ;  
13     ;  
14     ;  
15 174320 CPUFAL:   
16 174320 012700 000021 mov    CPUFAL,R0    ;R0 gets error code for P.roc failure  
17 174324 000543 br      CPUFAL    ;go to common error display routine  
18  
19 174326 J11000: ;  
20     ;-----  
21     ;  
22     ; Light the 'init' lamp to show J-11 test passed  
23     ;  
24     ;-----  
25     ;  
26  
27 174326 032737 000020 170042 bis    010010P,000040SR ;light the init lamp to show that J tests passed
```


MEMORY TESTS

```

1      .start MEMORY TESTS
2      ;+
3      ;-----
4      ;
5      ;       Test the first 2,048 bytes of Program Memory, and
6      ;       Find a working 8 Word partition in Program Memory
7      ;
8      ;inputs:  ROM Kernel PAR 5 = unit 0 (lower byte), type of boot (upper byte)
9      ;          ROM Kernel PAR 6 = CBR of Rx33 to use for boot
10     ;          ROM Kernel PAR 7 = validation word for unit and CBR parameters
11     ;
12     ;process: Find an 8 Word partition in Program Memory that passes a combined
13     ;          data integrity and addressing test. In addition to this 8 Word
14     ;          partition, the first 2048 bytes of Program memory must be working for
15     ;          use as the PDP-11 interrupt and trap vector area.
16     ;          If the first 2048 bytes do not work, the entire test
17     ;          is considered to have failed. If the first 2048 bytes are working,
18     ;          then try to find any 8 Word chunk of Program memory that works. If the
19     ;          first 2048 bytes are working, but no contiguous 8 Word partition can
20     ;          be found, the test fails.
21     ;
22     ;outputs: case 1 - no failure detected
23     ;          'Init' lamp is on, 'Fault' lamp is off
24     ;          00 = Base address of tested 8 Word partition
25     ;          ROM Kernel PAR's 5,6, and 7 are unchanged
26     ;          Control is passed to the Rx33 test
27     ;
28     ;          case 2 - failure detected (see above for definition of failure)
29     ;          'Fault' lamp is on
30     ;          'Init' lamp is on
31     ;          J-11 is in loop, monitoring the fault lamp
32     ;          Depressing the fault lamp causes blinking display of fault code.
33     ;          J-11 must be released to attempt recovery
34     ;
35     ;notes:
36     ;      1) the variable 'PARSIZ' defines the size of the partition to be tested
37     ;      2) the variable 'VECSIZ' defines the size of the vector area to test
38     ;
39     ;history of changes to 1"-1 tests:
40     ;
41     ;-----
42     ;-

```

..TEST 1 - Program Memory (Swap Bits)

```

1      .sbtcl ..TEST 1 - Program Memory (Swap Bits)
2      ;+
3      ;-----
4      ;
5      ;       Test the 'Swap Private/Control Memory Banks' bits in the P.1ec CSR
6      ;
7      ;inputs: none
8      ;
9      ;process: 1) set map bits
10     ;           2) Check that 'map PWB' is set
11     ;           3) Clear the 'map CWB' bit
12     ;           4) Check that the 'map PWB' bit is still set
13     ;           5) Clear the 'map PWB' bit
14     ;           6) Check that both map bits are clear
15     ;
16     ;outputs: Declare a P.1ec failure if any of the tests fail
17     ;
18     ;-----
19     ;-
20 174334 005267 176000      inc     TESTNO      ;set test number to 1
21 174340 012704 176040      mov     00P1ECR,R4      ;point R4 to P.1ec's CSR
22 174344 052714 006000      bis     00P0WB'00PWB,(R4) ;set both map bits
23 174350 032714 004000      bit     00P0WB,(R4)      ;check the map control mem bit
24 174354 001003            bne     30              ;br if bit is set
25 174356 042714 000000      40: bic     00P0WB'00PWB,(R4) ;try to clear both bits
26 174362 000736            br       0P1FAL          ;declare a P.1ec failure
27
28 174364 042714 004000      50: bic     00P0WB,(R4)      ;clear the map control mem bit
29 174370 032714 006000      bit     00P0WB'00PWB,(R4) ;test both bits (one should be set)
30 174374 001770            beq     40              ;br if neither bit is set
31 174376 042714 002000      bic     00PWB,(R4)       ;clear the map private bit (both clear now)
32 174402 032714 006000      bit     00P0WB'00PWB,(R4) ;check for both bits clear
33 174406 001363            bne     40              ;br if either bit failed to clear

```

..TEST 2 - Program Memory (Vector area)

```

1      .obj1 ..TEST 2 - Program Memory (Vector area)
2 174010 003067 175724      inc     TESTNO      ;set test number to 2
3 174014      NEXTST: ;
4
5
6      ;-----
7      ;
8      ;   Test the first 2048 bytes of program memory
9      ;
10     ;   If a failure occurs in the first 2048 bytes, then switch
11     ;   the memory bank that occupies that address range, if
12     ;   possible, and repeat test.  If the memory bank can not be
13     ;   switched, or if the second bank also fails, declare error.
14     ;-----
15 174014 003004      clr     R4      ;address 0 is base address of area to test
16 174016 012705 003776      mov     OVERSIZ-2,R5 ;R5 gets last address to test
17 174022 003003      clr     R3      ;R3 gets 0 of failing addresses to store (none)
18 174024 004767 002226      jsr     PC,MB/INJ ;call the memory test
19 174030 103012      bcc     NEXTST ;br if memory test passed
20 174032      NEXTST: ;
21
22     ;-----
23     ;
24     ;   if( 'bank swap'=0 ) then ( 'bank swap' gets a 1, retry test )
25     ;   if( 'bank swap'=1 ) then ( declare unrecoverable error )
26     ;
27     ; NOTE ON PROGRESSION OF 'SWP BANK' SIGNAL:
28     ;
29     ;   Pass 1 - 'swp bank' zero
30     ;   Pass 2 - 'swp bank' = one
31     ; note - now there is only one 'swp' possible (private memory banks)
32     ;-----
33 174032 012706 177352      mov     OVERSIZ,SP ;reset SP
34 174036 012703 170040      mov     OVERSIZ,R3 ;R3 points to P.roc Control and Status register
35
36 174042 032713 002000      ;-- If ----)
37                                bit     SWPFB,(R3) ;are banks swapped now?
38 174046 001070      ;-- Then ----)
39                                lsr     SWPFB ;br if banks are now swapped
40 174050 032713 002000      ;-- Else ----)
41 174054 000757      ;                                bit     SWPFB,(R3) ;assert 'swp banks' signal
42                                br       NEXTST ;retry memory test on first 2048 bytes

```

..TEST 3 - Program Memory (Partition area)

```

1      .sect1 ..TEST 3 - Program Memory (Partition area)
2 174036 PARTST: j+
3
4
5      Find a working 8 word partition in Program Memory
6
7      If no failures are detected, then continue with R33 test.
8      If a failure is detected, set the base address of the area
9      to be tested to the next address following the failure, and
10     try to find a contiguous 8 word area with no errors. If no
11     8 word area can be found, declare a memory test error.
12
13
14     |-----
15     |
16     |
17     |
18     |
19     |
20     |
21     |
22     |
23     |
24     |
25     |
26     |
27     |
28     |
29     |
30     |
31     |
32     |
33     |
34     |
35     |
36     |
37     |
38     |
39     |
40     |
41     |
42     |
43 174338 012706 004000 MEMERR: mov    0UECSIZ,SP      ;reset SP in case we trapped to here
44 174334 010004          mov    R0,R4      ;move failing address to R4
45 174336 062704 000002          add    R2,R4      ;adjust address past failure
46 174362 032704 000077          bit    R7,R4      ;is address on 32 word boundary?
47 174366 001004          beq    R0        ;br if it is
48 174370 042704 000077          bic    R7,R4      ;adjust address to 32 word boundary
49 174374 062704 000100          add    R4,R4      ;
50 174600 010405 100:      mov    R4,R5      ;set R5 to starting address
51 174602 062705 037776          add    0PARSIZ-2,R5 ;R5 now = last address of 8 word partition to test
52 174606 020527 160000          cmp    R5,0160000      ;see if we are into I/O page yet
53 174612 103006          bhis   BACKEND      ;br if we are into I/O page
54 174614 000732          br     MEMTRY      ;retry memory test on new partition
55
56 174616 010400          MEMERR: mov    R4,R0      ;R0 gets base address of 8 word partition
57 174620 010006          MOV    R0,SP      ; stack pointer now at base of tested mem

```

For Internal Use Only

MSC70 RSP Bootstrap 183-01 MICRO V05.03b Wednesday 06-Nov-85 11:12 Page 27-1

..TEST 3 - Program Memory (Partition area)

30 174622 062706 040000

30 174626 000401

00

01

ADD OPMSIZ,SP

RR R:RES

; stack now at TOP of tested area

; now start first Rx30 test

----->

1			
2			
3	174630		
4			
5			
6			
7			
8			
9			
10			
11	174630	012700	000022
12	174634		
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23	174634	106427	000340
24	174640	012701	170042
25	174644	042711	000017
26	174630	052711	000010
27	174634	005067	002520
28	174660	032711	004000
29	174664	001775	
30			
31			
32			
33			
34			
35			
36			
37	174666	042711	000037
38	174672	010011	
39	174674	012703	000007
40	174700	012702	045570
41	174704	077201	
42	174706	077304	
43			
44			
45			
46			
47			
48			
49			
50	174710	042711	000037
51	174714	012703	000005
52	174720	012702	045570
53	174724	077201	
54	174726	077304	
55	174730	000756	
56			
57			

```

.obj11  FAULT Lamp Routine
.enable L90
BACHED1: ;+
;-----
;
;   Declare a memory test error
;   Set the fault code register (R0) to 'Memory Module Failure'
;
;-----
;-
mov     000FAL,R0      ;R0 gets code for 'memory module failure'
BACHED1: ;+
;-----
;   Handle Fault lamp report for any error. R0 contains failure code
;
;   Clear all lamps except INIT (indicates whether J-11 tests passed).
;   Light the Fault lamp, then monitor the fault switch. When the Fault
;   switch is pressed, display a blinking failure code in the lamps.
;   This routine is stuck in the middle of the ROM so all can
;   get at it with a branch instruction.
;
;-----
;-
steps   0340           ;disable interrupts
mov     0054CSR,R1      ;point R1 to P.10c display control register
bic     0FLTUP*BLUP*HLUP*HOLUP,(R1) ;extinguish all lamps except INIT
bis     0FLTUP,(R1)     ;light the fault lamp
CLR     0XCSR           ;spin down the floppies. (If you 1001 here, bite it.)
400:    bit     0FLTSW,(R1) ;has the fault switch been depressed?
        beq     400      ;br if not
;+
;-----
;
;   Display the fault code in the lamps for 1/2 second
;
;-----
;-
500:    bic     0ALLUP,(R1) ;extinguish all lamps
        mov     R0,(R1)    ;put fault code in lamps
PTIME0: mov     07,R3      ;repeat 50 msec delay below 10 times
600:    mov     019320.,R2 ;R2 gets constant for 50 msec delay
700:    sub     R2,700      ;delay 50 msec
        sub     R2,600      ;delay 1/2 second (500 msec) total
;+
;-----
;
;   Extinguish all the lamps for 1/2 second
;
;-----
;-
bic     0ALLUP,(R1)      ;turn off all lamps
PTIME1: mov     05,R3      ;repeat 50 msec delay below 10 times
800:    mov     019320.,R2 ;R2 gets constant for 50 msec delay
900:    sub     R2,900      ;delay 50 msec
        sub     R2,800      ;delay 1/2 sec (500 msec) total
        br      500        ;turn on the lamps for 1/2 second again
.disable L90
;+-----+

```


For Internal Use Only
HBC70 ROM Bootstrap X03-01 MICRO V05.03b Wednesday 06-Nov-85 11:12 Page 20-1
FALTY Lamp Routine

30
30

```

1      .obj1 ..TEST 4 - Rx33 Controller Hardware Test(s)
2
3      ;-----
4      ; ROUTINE DESCRIPTION :
5      ; This is the first test for the Rx33 subsystem.
6      ; First it is determined if there is an Rx33 controller responding at
7      ; address 177777400. If so, registers 177777402-406 are tested for stuck bits.
8      ; If this test fails, the register being accessed and found in error is placed
9      ; in the error table at memory address 400.
10
11      ; NOTE : Bits 0:7 of the Rx33 CSR (177777400) cannot be written since
12      ; this is the command register, and we don't want to start the Rx doing some
13      ; command.
14
15      ; INPUT PARAMETERS :R0 has base address of partition just tested by memory.
16      ; SP is set to top of this partition.
17      ; OUTPUT PARAMETERS : If test fails control transfers to fault light routine.
18      ; On success, execution falls thru to further Rx33 tests.
19      ;-----
20      ; Register Assignments
21      ;
22      ; R0 : Base address of tested partition from memory program
23      ; R1 : data image to write/compare against
24      ; R2 : used for address of register under test
25      ; R3 : scratchpad, used for timeout
26      ; R4 : number of registers to test
27      ; R5 : data image count
28      ; R6 :
29
30      ;-----
31      ;
32      ; R0:1 INC TESTNO ; set test number to 4
33
34      ; R0:2 MOV R4,R2 ; SET ON ROM VECTOR, in rom
35      ; R0:3 MOV OR00001,(R2)+ ; AND DISABLE INTERRUPTS, if ROM occurs
36
37      ; If you get here, there is an Rx33 out there, otherwise you would ROM trap.
38      ; Whether or not the Rx is healthy yet is open for debate at this point.
39
40      ; R0:4 MOV OR0CSR+2,R2 ; START OUT IN 'TRACK' REGISTER
41      ; R0:5 MOV R2,R4 ; NUMBER OF REGISTERS TO DO
42
43      ;-- Begin ---)
44      ; R0:6 MOV R16,R5 ; NUMBER OF DATA PATTERNS TO TEST
45      ; R0:7 MOV R1,R1 ; STARTING IMAGE
46
47      ; R0:8 MOV R1,R2 ; IMAGE--> REGISTER
48      ; R0:9 CALL WAIT ; wait gives you 33.2 us of slack time
49      ; R0:10 CMP R2,R1 ; COMPARE WAS WRITTEN WITH WHAT IS
50      ; R0:11 BNE BITSTK ; go to bit stuck report----> exit
51
52      ; R0:12 ASL R1 ;
53      ; R0:13 SOB R5,R5 ; do this until we hit bit 15
54
55      ;-- End ---->
56
57      ; R0:14 MOV (R2)+,R5 ; ONE INSTRUCTION FOR INC R2 BY TWO
58      ; R0:15 SOB R4,R4 ; DO THIS FOR THREE REGISTERS
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

For Internal Use Only

IBC70 ROM Bootstrap X83-01 MACRO V05.03b Wednesday 06-Nov-85 11:12 Page 29-1

..TEST 4 - Rx33 Controller Hardware Test(s)

```
30                                     ; now all general registers are tested. Verify some bits in the upper byte
31                                     ; of the CCR to make sure we have COMPLETE CONTROL.
32
33                                     MOV     @PTRREG@!DISQ!verify,R1 ; image of what should be
34 01 173014 012701 011004                                     MOV     @CCR,R2 ; address of RX controller
35 02 173020 012702 177400                                     BIT     @160000,(R2) ; make sure no bits are stuck
36 03 173024 032712 160000                                     BNE     BITSTK ; EXIT IF STUCK BITS
37
38                                     ; when you fall thru to the next page general health of the Rx33 registers is
39                                     ; indicated. Proceed with caution, and TERMINATE ON ERROR!!
40                                     ;----->
```

166

..TEST 4 PART II Interrupts & DMA

```
1      .obj1 ..TEST 4 PART II Interrupts & DMA
2      |-----|
3      | ROUTINE DESCRIPTION :
4      | This test is to see whether the 8x33 can perform basic operations such
5      | as a single interrupt. DMA hardware is tested to see that address counters
6      | increment properly, and that the data path to/from memory is error free.
7      | All of the data path bits are checked , as well as the parity.
8      |
9      | INPUT PARAMETERS :
10     | OUTPUT PARAMETERS :
11     |-----|
12     | Register Assignments
13     |
14     | R0 :
15     | R1 : BIT mask to load DMA address
16     | R2 : Address of register under test
17     | R3 :
18     | R4 :
19     | R5 :
20     |-----|
```

For Internal Use Only

HEC70 ROM Bootstrap X83-01 MACRO V85.03b Wednesday 06-Nov-85 11:12 Page 31

..TEST 4 PART II Interrupts & DMA

1	175032	010201		MACRO: MOV	R2,R1	; R2 STILL HAS CSR ADDRESS (also data we want to use)	
2	175034	005722		TST	(R2)+		
3	175036	010122		MOV	R1,(R2)+	; move 177000 to R1TRK	
4	175040	012767	176572	003162	MOV	000FIN,R1TRK	; place the interrupt vector for forced interrupt
5	175046	012767	000340	003156	MOV	0340,R1TRK+2	; move priority during interrupt
6	175054	106427	000000		HTPS	00	; set priority to zero so we can be interrupted
7	175060	010122		MOV	R1,(R2)+	; Move same thing to R1SEC	
8							
9	175062	004767	001512		CALL	WAIT	; okay so it takes two extra words... it's safe
10							
11	175066	012785	000003		MOV	03,R1	; basic error code in case of failure to interrupt
12	175072	012783	031330		MOV	0000H!DISRQ!FACINT,R1	; R1 has command to be able to force interrupt
13	175076	004767	001506		CALL	EXECM	; go and execute the interrupt command
14	175102	103543			BCS	T4ER1	; error = controller did not interrupt
15							
16	175104	012767	011320	002266	MOV	0000H!DISRQ!RSINT,R1CSR	; reset after forced interrupt
17	175112	004767	001462		CALL	WAIT	
18	175116	012767	176566	003104	MOV	000INS,R1TRK	; place regular interrupt handler
19	175124	012712	040000		MOV	000ATST,(R2)	; load test mode to R2TRK
20	175130	004767	001444		CALL	WAIT	; WAIT FOR 33.2 us
21	175134	012703	021254		MOV	0000H!WRSEC,R1	; this will be the WRITE command
22	175140	032767	000200	002232	BIT	000000,R1CSR	; verify that we do not have ready do to DMA mode
23	175146	001520			BEQ	T4ER2	; (DMA MODE DID NOT ISOLATE DRIVES)
24							
25	175150	004767	001434		CALL	EXECM	
26	175154	103516			BCS	T4ER1	; no interrupt
27							
28	175156	032712	000400		BIT	0bit8,(R2)	; see if we got the carry
29	175162	001511			BEQ	T4ER3	; counters wrong (DMA logic bad)
30							
31	175164	030142			BIT	R1,-(R2)	; check for any upper order bits that shouldn't be
32	175166	001107			BEQ	T4ER3	; counters wrong (DMA logic bad)
33							
34	175170	032742	177000		BIT	0177000,-(R2)	; check for any bits stuck in R1TRK
35	175174	001104			BEQ	T4ER3	; bits stuck in count (DMA LOGIC BAD)

For Internal Use Only

MC70 RM Bootstrap X03-01 MC70 V05.03b Wednesday 06-Mar-85 11:12 Page 32

.TEST 4 PART II Interrupts & DMA

```

1
2
3
4 175176 005067 002204 CLR R000R ; remove DMA mode so we can load other registers
5 175202 004767 001372 CALL WAIT
6 175206 005022 CLR (R2)+ ; zero the track register too
7 175210 004767 001364 CALL WAIT
8 175214 012722 000125 MOV 0125,(R2)+ ; load sector w/data pattern; update pointer to R000R
9 175220 004767 001354 CALL WAIT
10 175224 012712 040000 MOV 000ATST,(R2) ; set dma test mode in R000R
11 175230 012703 021214 MOV 000DM!RDEC,R3 ; now do DMA read from disk to MEMORY
12
13 ; now at this point if you get a PARITY ERROR when you check the result in
14 ; memory, you will trap thru 114 to the test error vector below.
15 ; Then you will end up with a fault light. If you get a parity error after the DMA
16 ; operation, there is no sense in recovery.
17
18 175234 012737 175402 000114 MOV 0T4ER5,00114 ; set up parity trap vector for checking
19 175242 004767 001342 CALL EXECBN ; do the command ( should write memory)
20 175246 103461 BCS T4ER1 ; (INTERUPT DID NOT OCCUR)
21
22 175250 005012 CLR (R2) ; Remove DMA test mode; test result so registers settle
23 175252 022737 052525 000000 CVP 052525,000 ; see if data transferred (parity trap may take you here)
24 175260 001031 BIE T4ER4 ; (DMA DATAPATH BAD)
25
26 175262 012742 000031 MOV 031,-(R2) ; load pattern which will have different parity
27 175266 004767 001306 CALL WAIT
28 175272 012767 040000 002106 MOV 000ATST,R000R ; set DMA TEST MODE
29 175280 012703 021214 MOV 000DM!RDEC,R3 ; load read command
30 175284 005712 tst (r2) ; pad the loop
31 175286 004767 001276 CALL EXECBN ; should have settled by now
32 175312 103437 BCS T4ER1 ; no interrupt, (op not complete)
33
34 175314 022737 014431 000002 CVP 014431,002 ; see if two has pattern
35 175322 001030 BIE T4ER4 ; if data not correct
36
37 175324 012737 174630 000004 MOV 000DMEM,004 ; move memory test vector back
38 175332 012737 174630 000114 MOV 000DMEM,00114 ; if we get parity error after this.. bad memory
39 175340 000430 BR R00T10 ; hardware is OK; find the first working drive
40
41

```


Register & hardware test error traps

```

1      .sect1 Register & hardware test error traps
2      |-----|
3
4 173342      R00000:
5
6              ; 1001 hardware vector is set to here in case of a 1001 when trying to get at
7              ; the R23 registers.
8              ; save register being addressed and 1001 code in R2 error table; GOTO lights.
9              ; we will leave the 1001 vector set to here during the hardware test as well.
10             ; we might catch intermittent failures that way, and it's free.
11
12 173342 005302      DEC     R2
13 173344 005302      DEC     R2              ; ADJUST R2, SINCE AUTO INC IS USED
14 173346 010267 003030      MOV     R2,(ERTAB0+2)      ; save the address which gave 1001
15 173352 012767 000001 003020      MOV     01,ERTAB0      ; and deposit '1001' when accessing R23' code
16
17 173360 000416      BR      R00000      ; GOTO fault light; that's all we know
18
19 173362      BITSTK:
20              ; if we find bad bits in the R23 registers we end up here
21              ; No stuff the bad bit info as well as the address involved
22
23 173362 012704 000400      MOV     0ERTAB0,R4      ; ADDRESS OF TABLE
24 173366 012724 000002      MOV     02,(R4)+      ; set code = bad bit in register
25 173372 010224      MOV     R2,(R4)+      ; SAVE OFFENDING REGISTER
26 173374 010124      MOV     R1,(R4)+      ; EXPECTED BIT PATTERN
27 173376 011214      MOV     (R2),(R4)      ; AND WHAT WAS REALLY THERE
28 173400 000406      BR      R00000      ; goto fault light routine
29
30              ; This completes data entry into the error table. Xfer to common R23 error point.
31
32      |-----|
33      ; Below are the error traps for the BPA/INTERRUPT hardware tests.
34      ; Depending on your entry point, by the time you get to the move of R5 to the error
35      ; table in low memory, you will have incremented a number of times (0:4) which
36      ; will give you some unique error code word in the table. These are all hardware
37      ; faults of a fatal nature which would prevent booting by the R23.
38      |-----|
39
40 173402 005205      T4ER5: INC     R5              ; ERROR = 7      Parity error in datapath
41 173404 005205      T4ER4: INC     R5              ; ERROR = 6      Data wrong in memory
42 173406 005205      T4ER3: INC     R5              ; ERROR = 5      BPA address counters wrong
43 173410 005205      T4ER2: INC     R5              ; ERROR = 4      Test mode hardware fault
44 173412 010567 002762      T4ER1: MOV     R5,ERTAB0      ; ERROR = 3      Interrupt failed
45
46              ; just fall thru then to common gag point below.
47 173416      R00000:
48
49              ; the correct error information is already assumed to have been
50              ; stuffed in the error table.
51 173416 000167 177206      JWP     R00000      ; put a normal code in R2; goto common fault
52
53      |-----|
54

```

..TEST 6 - Find a good Rk33 drive.

```

1      .subtl ..TEST 6 - Find a good Rk33 drive
2      ;+
3      ;~~~~~
4      ;
5      ;inputs:  R0 = Base address of the 8 Kword partition(supplied by memory test)
6      ;          R0R1 Kernel PAR 5 = Unit 0 of Rk33(lower byte) and type of boot (upper bit)
7      ;          R0R1 Kernel PAR 6 = CSR address of Rk33 to use for boot (16 bits)
8      ;          R0R1 Kernel PAR 7 = Validation word for unit and CSR
9      ;
10     ;
11     ;process: Find a working Rk33 floppy drive can be used to read in the Init P.roc test.
12     ;          A check is made to see if we were passed a specific Rk33 CSR and unit to use.
13     ;          Parameters in R0R1 Kernel PAR's 5, 6 and 7 are checked for validity by testing
14     ;          the validation word is equal to the XOR of the Unit & CSR words. If parameters
15     ;          are valid, the load is attempted via the Rk33 specified. If parameters in the
16     ;          PARs are not valid, or if load attempt fails on unit specified, the default
17     ;          selection sequence (described below) is used.
18     ;
19     ;          DEFAULT SELECTION SEQUENCE (it's default of the drive)
20     ;          Either of the Rk33 drives that can be connected to the Mncd2 can be used as a
21     ;          boot device. Testing begins with Rk33 unit 0. If unit 0 does not
22     ;          respond, unit 1 is tried. If unit 1 also fails, we are in trouble.
23     ;
24     ;          Once we have a drive that we think works, we use 'R0READ' within the ROM
25     ;          to load, or try to load, the first 8 blocks to the 8 Kword partition in
26     ;          Program memory. If the first word of the loaded image is a NOP instruction,
27     ;          control is passed to the first instruction of the loaded image. If the
28     ;          first instruction is not a NOP, then the next Rk33 unit in sequence is tried.
29     ;
30     ;          Failure to find any working Rk33 drive with media mounted is a fatal
31     ;          error and results in the failure action listed below (see 'outputs').
32     ;
33     ;outputs: Case 01 - A working Rk33 with bootable media mounted was found:
34     ;          Parameters passed:
35     ;          a) Type of boot (warm or cold)
36     ;          b) Base address of tested 8 Kword partition
37     ;          c) Unit number of the Rk33 that was used to load
38     ;          d) CSR address of the Rk33 that was used to load
39     ;
40     ;          Case 02 - No working Rk33 could be found:
41     ;
42     ;          State of J-11:
43     ;          a) 'Init' lamp is lit
44     ;          b) 'Fault' lamp is lit
45     ;          c) J-11 in loop monitoring the 'fault lamp'. Depressing
46     ;             the 'fault lamp' results in a blinking display of the
47     ;             fault code for 'Load failure'
48     ;          d) J-11 must be rebooted to attempt recovery
49     ;
50     ;          Probable failure:
51     ;          a) No MBC70 system diskette is mounted
52     ;          b) Faulty Rk33 unit (if boot fails on just one unit)
53     ;          c) Faulty Rk33 controller (if boot fails on both units)
54     ;          d) Gamma rays
55     ;
56     ;          Recovery action:
57     ;          a) Mount an MBC70 system floppy (if none was mounted) and reboot

```

..TEST 6 - Find a good R2D drive.

2010年12月10日

```

;
; b) Switch the HBC70 system floppy to the other Rx33 unit
;      (if proper media was mounted)
;
; c) Refer to description of Rx33 error table in the USEFUL INFORMATION
;      section of this listing
;
; d) Kick system, being careful not to injure you foot
;
;Notes:
;
;History of changes to this test:
; Feb 1985 : changes to boot from Rx33. CSR to use left in for compatibility
; Mar 1985 : more changes, and register test routine split out as separate test
; ~~~~~
;

```

..TEST 6 - Find a good Rx33 drive.

```

1 175422                                RCTYFud:
2 175422 005267 174712                inc     TESTNO          ; set test number to 6
3 175426 012767 100000 001752        mov     ONSTDR,CONR      ; set green LED on board; controller hardware good
4                                     ;
5                                     ;
6                                     ;
7                                     ;
8                                     ;
9                                     ;
10                                    ;
11                                    ;
12                                    ;
13 175434 010006                        mov     R0,SP          ;point stack to lowest address of BK partition
14 175436 062706 000000                add     ONP0S1Z,SP       ;adjust to last address (+2) of BK partition
15                                    ;
16                                    ;
17                                    ;
18                                    ;
19                                    ;
20                                    ;
21                                    ;
22                                    ;
23                                    ;
24                                    ;
25                                    ;
26                                    ;
27 175442 016704 174704                mov     ONP0R5,R4        ;R4 gets the unit word (upper bit=type of boot)
28 175446 042704 100000                bic     0100000,R4       ;clear out type of boot field
29 175452 016702 174676                mov     ONP0R6,R2        ;R2 gets the CSR word
30 175456 074002                        xor     R4,R2          ;R2 gets XOR of unit and CSR words
31 175460 020267 174672                cmp     R2,ONP0R7       ;is specified unit and CSR valid?
32 175464 001025                        bne     RDEF          ;br if not and use default sequence
33
34 175466 042704 177000                bic     0177000,R4       ;clear out upper byte of unit word
35 175472 016701 174656                mov     ONP0R6,R1        ;R1 gets CSR address for Rx33 read subroutine use
36 175476 020127 100000                cmp     R1,0160000      ;ensure CSR address in I/O page
37 175502 103416                        blo     RDEF          ;br if CSR is not in I/O page, use default sequence
38
39 175504 020427 000001                cmp     R4,01          ;ensure unit 0 is not greater than 1
40 175510 003013                        bgt     RDEF          ;br if unit 0 is greater than 1, use default seq
41
42 175512 004767 000204                call    RDEFST         ; See if requested drive can recalibrate and verify
43 175516 103003                        bcc     10              ; if test passed, try to boot from unit
44
45 175520 004767 000100                call    ERRSNW         ; otherwise save the reason the default unit failed
46 175524 000405                        br     RDEF          ; then use default sequence
47
48 175526 004767 000304                10:  call    READBT         ;read boot blocks and check for bootable image
49 175532 103037                        bcc     transr         ;br if read ok, and bootable image loaded
50
51 175534 004767 000064                call    ERRSNW         ; save the failure to boot
52
53                                    ; fall thru and try default selection sequence
54
55

```

..TEST 6 - Find a good R20 drive.

```

1
2          ;*****  DEFAULT BOOT SEQUENCE  *****
3
4 173540      CODE7:
5
6 173540 003004      CLR    R4          ;set unit 0 to zero
7
8 173542 012701 177000 10:  MOV    @R0CSR,R1      ; move address of CSR
9 173546 004767 000130      CALL  R0RST          ; Go and attempt to do a recal/verify
10 173552 103007          bcc     Z0              ; br if successful, to boot
11
12 173554 004767 000044      CALL  ERRSAW          ; else, call error_save. R3=error code R4=unit number
13 173560 005204          INC     R4              ; next try unit+1
14 173562 020427 000001      CMP     R4,#1          ; for unit=0, should be minus, for unit=1 should be 0
15 173566 101763          BLOS    Z0              ; go back thru and attempt to recalibrate drive one
16
17 173570 000411          BR     R0FAL          ; R0FAL is common exit point if no drives worked
18                                     ; Only this location can give a '23' failure
19
20                                     ;-----
21                                     ;
22                                     ; First try to read a bootable image from unit 0 of the
23                                     ; currently selected controller. If we can't load a bootable
24                                     ; image from unit 0, try unit 1.
25                                     ;
26                                     ; (If the read succeeds, we have read the boot blocks and the
27                                     ; disk directory into the 0 fixed partition.)
28                                     ;
29                                     ;-----
30                                     ;
31
32 173572 004767 000240 20:  CALL    READBT          ; read boot blocks and check for bootable image
33 173576 103015          bcc     TIMEFR          ; br if read successful and image is bootable
34
35 173600 004767 000020      CALL  ERRSAW          ; Otherwise save error code (13:17)
36
37 173604 005204          INC     R4              ; point to next unit number
38 173606 020427 000001      CMP     R4,#1          ; see if we have exceeded the number of drives
39 173612 101753          BLOS    Z0              ; go back and execute test string for new drive
40
41 173614 012700 000023  R0FAL: MOV    @LOADFL,R0      ; failure code for boot device failure
42 173620 000167 177010      JF     R0FAL          ; goto common error routine
43
44 173624 110564 000410  ERRSAW: MOV    R5,ERTAB6(R4)      ; put in correct byte
45 173630 000207          RETURN          ; return to caller (saves code)
46
47

```


..TEST 7 - Transfer Control to Loaded Image

```

1      .cbl1 ..TEST 7 - Transfer Control to Loaded Image
2      ;+
3      ;-----
4      ;
5      ; The image just loaded to Program Memory is either an Init P.IOC test
6      ; or a non-HBC70 bootstrap (e.g., RT-11, RSK-11). Either way, the image
7      ; must be massaged to work properly. In the case of 'Init P.IOC' test,
8      ; a one block (512 byte) 'hole' exists between the first and second blocks
9      ; of the test, due to presence of a 'Volume ID' block on the Rk33. This
10     ; hole is collapsed before transferring control to the loaded image. In
11     ; case of a foreign (non-HBC70) bootstrap, the image must be moved from
12     ; the tested 8 word partition down to address 000000 to work properly.
13     ;
14     ; After 'massaging' the loaded image, set up the parameters to be
15     ; transfer to the image. For Init P.IOC test, transfer to the base address
16     ; of the loaded image. For a foreign bootstrap, transfer control to location 0.
17     ;
18     ;inputs:      All tests passed (implied)
19     ;              Rk33 parameters (CNR, Unit number)
20     ;              Base address of working 8 word partition
21     ;              Type of boot (warm or cold)
22     ;              'Init' lamp is lit
23     ;              'Fault' lamp is off
24     ;
25     ;process:     Check the second instruction of the image just loaded
26     ;
27     ;              if ( second instruction = NBP ) then ( Init P.IOC test was loaded )
28     ;              else ( a foreign bootstrap was loaded )
29     ;
30     ;              if ( Init P.IOC test was loaded )
31     ;                  transfer to first address loaded)
32     ;
33     ;              if ( foreign boot was loaded )
34     ;                  then ( move image down to location 0, transfer to loc 0 )
35     ;
36     ;outputs:     See 'inputs' above
37     ;
38     ;notes:
39     ;
40     ;History of changes to this routine:
41     ; 4-Mar-85 compression no longer needed for Rk33, so this was removed
42     ;-----
43     ;-

```



```

1 175632                                TMRFR: j+
2
3
4
5                                     Set up the parameters that specify the RX33 unit and CSR used for
6                                     the load just completed. They may have changed if the unit
7                                     specified for booting was not working.
8
9
10 175632 005267 174502                j-
11 175636 042737 077777 172352        inc    TESTMD      ;set test 0 to 7
12 175644 110437 172352                bic    077777,000KPAR5 ;mask everything except type of boot bit
13 175650 010137 172354                movb   R4,000KPAR5  ;save unit 0 we booted from
14                                     mov    R1,000KPAR6  ;save CSR of R233 we booted from
15
16                                     j+
17
18                                     Determine if we loaded the 'Init P.IOC' test or a foreign bootstrap
19                                     if (second instruction of loaded image = NOP)
20                                     then ( Init P.IOC test was loaded)
21                                     else ( foreign boot was loaded )
22
23
24 175654 010002                        j-
25 175656 026227 000002 000240 100:    mov    R0,R2      ;R2 gets base address of 0 word partition
26 175664 001005                        cmp     2(R2),0000240 ;is second instruction a NOP?
27                                     bne     500          ;br if not, must be a foreign bootstrap
28
29                                     j+
30
31                                     We have loaded the 'Init P.IOC' test
32                                     The 7 blocks loaded are as follows:
33
34                                     block 0      first 512 bytes of 'Init P.ioc' test
35                                     block 1      Not loaded
36                                     block 2      second 512 bytes of 'Init P.ioc' test
37                                     block 3      third 512 bytes of 'Init P.ioc' test
38                                     block 4      fourth 512 bytes of 'Init P.ioc' test
39                                     block 5      fifth 512 bytes of 'Init P.ioc' test
40                                     block 6      first half of first RT-11 directory segment
41                                     block 7      second half of first RT-11 directory segment
42

```

..TEST 7 - Transfer Control to Loaded Image

```

1
2
3
4
5
6
7
8
9
10
11
12 175666 016700 174460
13 175672 016701 174436
14 175676 000112
15
16
17
18
19
20
21
22
23
24
25
26
27 175700 010203
28 175702 003004
29 175704 012705 000400
30 175710 012324
31 175712 077302
32
33
34
35
36
37
38
39 175714 116700 174432
40 175720 003007
41

```

```

;-----
;
; Set up parameters to be passed to the 'INIT P.IOC' test
;
; R0 = Unit 0 of Rn33 used to boot, and 'type of boot'
; R1 = CBR address of Rn33 (preserved for compatibility)
; R2 = Base address of the tested 8 word partition
; Priority is set to 7 for entry into just-loaded image
;-----
;
; R0 gets unit 0 and type of boot
; R1 gets CBR address of Rn33
; transfer to the Init P.ioc test
;-----
;
; The image just loaded is a foreign (non-MBC70) bootstrap. It was
; loaded to the 8 word partition in Program memory, but designed to
; execute from locations 000000 thru 000776. So we move the first block
; from the 8 word partition, down to address 000000, before jumping
; to address 000000 to begin the bootstrap.
;-----
;
; R3 gets base address of load
; R4 points to where to start moving data
; set R5 to count 1 block of data
; move a word
; branch until block is moved
;-----
;
; Now jump to location 0 to continue foreign bootstrap
;-----
;
; R0 gets unit number word
; jump to 0 to start foreign bootstrap
;-----

```

R0RST- perform a RECAL/VERIFY operation.

```

1      .sbrl R0RST- perform a RECAL/VERIFY operation
2
3      ;+
4      ;~~~~~
5      ;
6      ;      Test intended unit number of Rx for ready/and track 0.
7      ; This is used as a test for the init code in the boot rom, as well as
8      ; a reposition routine every time a drive is accessed in the offline diagnostics.
9      ; It sort of replaces the old routine T505YC, which synchronized with the TUS0.
10     ; 15-Oct-1985 we use the fact that interrupts have been tested already to check
11     ; to do this RECAL command. Since this routine can be called from outside the Boot
12     ; ROM we 'steal' the interrupt vector and then replace it on exit.
13     ; The verify flag is set in addition to RECAL command, so the track register
14     ; gets loaded with the correct track at the end of the command.
15     ; If the drive is already at track 0 this routine takes no time at all. Otherwise,
16     ; it can take up to 260ms to pull in from the outermost cylinder. We allow 833 in
17     ; the command execute routine.
18     ;
19     ;inputs:
20     ;      R0 = Base address of tested partition in Program Memory
21     ;      R1 = CSR address of RX to use, passed for compatibility
22     ;      R4 = Unit number to test. (0 OR 1 in low byte)
23     ;      R2, R3, R5 are SCRATCH
24     ;
25     ;process:
26     ;      Steal INTERRUPT VECTOR for Rx23
27     ;      Give a RECALIBRATE to the drive
28     ;      Test for READY and NOT_BUSY from selected Rx drive
29     ;      Test for track 0. If not track 0, error out
30     ;      RETURN to calling program
31     ;
32     ;outputs:
33     ;      R0, R1, R4 are PRESERVED
34     ;      R2, R3 and R5 are Clobbered
35     ;      CARRY is set if ERROR or TIMEOUT occurred
36     ;      R5 will have a valid error code if CARRY is SET, else zero
37     ;~~~~~
38     ;-

```

REQUEST- perform a RECAL/VERIFY operation.

```

1 175722                                REQUEST:
2
3 175722                                PUSH    RXIVEC                ; save current interrupt vector
4 175726                                PUSH    ROPBM                ; save priority
5 175730 012767 176366 002270          MOV     00KINH,RXIVEC        ; replace with ours
6 175740 106427 000000                  MTPS     00                ; set priority to zero now
7 175744 012705 000010                  MOV     010,R5                ; basic error code, device not ready
8 175750 012703 001320                  MOV     00TENDM:RSINT,R3        ; zero/select the drive
9 175754 004767 000630                  CALL    EXECOM                ; load command, expect timeout
10
11 175760 012703 031014                  MOV     0000M:DISQ:RECAL:VERIFY,R3    ; the recal command to do
12 175764 004767 000630                  CALL    EXECOM                ; do the recal operation
13
14 175770 016703 001404                  MOV     01CSR,R3                ; get current image of CSR
15 175774 032703 000200                  BIT     0000RDY,R3            ; see if Not-ready is set
16 176000 001010                        BNE     T30r1                ; door open, or no floppy (ERROR 10)
17
18 176002 032703 000030                  BIT     000FERR:CRCDERR,R3        ; check for RNF -or- CRC errors
19 176006 001004                        BNE     T30r2                ; (ERROR 11)
20
21 176010 032703 000004                  BIT     01TRACK0,R3            ; see if track_0 is set
22 176014 001003                        BNE     T30k1T                ; if not, fall thru to (ERROR 12)
23
24 176016 005205                        T30r3: INC     R5                ; ERROR = 12 TRACK_0 NOT SET
25 176020 005205                        T30r2: INC     R5                ; ERROR = 11 CRC/RNF ON RECAL/VERIFY
26 176022                                T30r1:                        ; ERROR = 10 DEVICE NOT READY
27 176022 000261                        SEC                                ; pass R5 to next level of software to process
28
29 176024                                (Exit: POP     ROPBM                ; restore PBM. Preserve for external calling
30 176030                                POP     RXIVEC                ; replace interrupt vector
31 176034 000207                        RETURN                ; and return VIA here
32
33

```

```

1      .subtl Read Boot Blocks to Memory
2
3      ;-----
4
5      ; Read R03 boot blocks to Program Memory and test for Bootable image
6
7      ;inputs:
8      ; R0 = Base address of tested Program Memory partition
9      ; R1 = CDR address of R03 to be used
10     ; R2 = SCRATCH
11     ; R3 = SCRATCH
12     ; R4 = Unit # of R03 to be used
13     ; R5 = The LBN to read.
14
15     ;process:
16     ; Issue a R033 read command for 1 block, load the data beginning at
17     ; the address in R0.
18     ; If that succeeded, issue a command to read blocks 2:8.
19     ; NOTE: Of the 8 blocks read, 5 are boot blocks, 1 is volume ID block,
20     ;       2 are directory blocks.
21
22     ;outputs:
23     ; C clear if read is successful (no errors and first instruction = NOP)
24     ; C set if read fails or first instruction is not a NOP
25     ; R0,R1, and R4 are PRESERVED
26     ; R2 and R3 are Clobbered
27     ; R5 has failure code, if applicable
28
29     ;-----
30     ;~
31 170036 READBT: ;+
32     ;-----
33     ;
34     ; call the R033 loader/read routine, (R0READ)
35     ; This is the same routine that is called by the loader (later)
36     ;-----
37     ;~
38 170036 PUSH    R0          ;save base address of partition on stack
39 170040 MOV     R0,R2       ;R2 gets starting address for load
40 170042 MOV     R4,R0       ;R0 gets unit number of R033
41 170044 MOV     R1,R4       ;R4 gets number to load first
42 170050 CLR     R5         ;R5 gets block number of 0
43
44     ;call the read subroutine. R2 has updated VA
45     ; R0,R1,R4 are untouched at return
46 170052 CALL    R0READ
47 170056 BCS     100        ; branch if carry... failed first read
48
49 170060 MOV     R2,R5       ; otherwise get rest starting at block two
50 170064 MOV     R6,R4       ; R4 gets number of blocks to load following 0
51 170070 CALL    R0READ     ; and get the remaining XXXX blocks: of 1..10:0
52 170074 BCS     100        ; if error on the transfer, then return
53

```

1					
2					
3					
4					
5					
6					
7					
8					
9					
10	176076	010004	MOV	R0,R4	; restore unit number data to R4
11	176100		POP	R0	; Get old base address from stack
12	176102	021027 000240	cmp	(R0),0000240	; is first instruction of image loaded a NOP?
13	176106	001407	BEQ	200	; if so, then image 'should' be bootable
14					
15	176110	012705 000017	MOV	#17,R5	; error code for non-bootable image load
16	176114	000402	BR	110	; punt out of here; set carry on the way out
17					
18	176116	010004	100: MOV	R0,R4	; if this path taken, return with carry set
19	176120		POP	R0	; and the old base address restored, just like
20					
21	176122	000261	110: SEC		; the above code. R5 should have error code
22	176124	000207	RETURN		; exit on error location
23					
24	176126	000241	200: CLC		
25	176130	000207	RETURN		; RETURN with no errors. Ready to start Boot
26					
27					
28					

DREAD- read disk blocks to memory.

```

1      .start DREAD- read disk blocks to memory
2
3      -----
4      ROUTINE DESCRIPTION :
5      DREAD is the heart and soul of the bootstrap operation.
6      Have we attempt to load from the Rn, according to the parameters we are
7      passed in the general registers. This routine is also callable by the world
8      at large to permit further loading by INPIO or OLP10.
9
10     29-Oct-85 : add clear of all but low bit of R0 ( unit 0 on entry)
11     -> This is for safety on a warm reboot
12
13     INPUT PARAMETERS : Unit number, Virtual address to load to,
14                       0 of LBNs to do, and lbn to start from
15     OUTPUT PARAMETERS : R0 restored. R5 has completion code
16                       R2 has updated Virtual address at end
17                       C bit clear if transfer successful
18                       C bit set if not. R5 has error code
19     -----
20
21     Register Assignments
22
23     R0 : UNIT 0 (passed by caller)
24     R1 :
25     R2 : Virtual address to transfer data to (passed by caller)
26     R3 :
27     R4 : Number of LBNs to transfer, (passed by caller)
28     R5 : LBN to start at (passed by caller) (error code on return)
29     -----
30
31 DREAD::
32     PUSH    RXIVEC          ; save old interrupt vector
33     PUSH    RXPSI          ; save the PSI
34     MOV     @RXINS, RXIVEC  ; stuff address of boot ROM's interrupt handler
35     BIC     @177776, R0     ; assure unit number okay on warm reboots
36
37     PUSH    R0              ; SAVE ALL THESE
38     PUSH    R1
39     PUSH    R4              ; save these
40
41     HTPS    @0              ; set priority to zero
42     MOV     R4, R3          ; play register game here
43     MOV     R3, R4          ; now R4 has unit, as it should
44     MOV     R3, R0          ; and R0 has number of LBNs to do
45
46     ;-- Begin --- MAXIMUM OUTSIDE OF READ LOOP
47     RXFLP: CALL    RXJGAT    ; sets up the Rn33 registers
48     PUSH    R5              ; save R5 (lbn) use internal as error code
49     MOV     @13, R5         ; base error code internally
50     BCC     @OLP1          ; go ahead with transfer if seek not needed

```

For Internal Use Only

MAC70 IBM Bootstrap X03-01 MACRO V05.03b Wednesday 06-Nov-85 11:12 Page 45

ROREAD- read disk blocks to memory.

```
1
2
3
4
5
6
7
8
9
10 176210 012703 021024      DOSEEK: MOV      0000H/SEEK/Verify,R3      ; Create command for seek
11 176214 004767 000370      CALL      EXECOM      ; call the common command execute routine
12 176220 103037              BCS      R0ER1      ; otherwise read seek error
13
14 176222 032767 000231 001150  P000H: BIT      000000V/00F0R/00C0R/0000Y,R0C0R      ; any errors during seek?
15 176230 001032              BNE      R0ER2      ; If so this is the 4th distinct error
16                          ; If there were no errors, then proceed on thru with the read routine
```

NBC70 80400 Bootstrap X83-01 HRC80 V05.03b Wednesday 06-Nov-05 11:12 Page 46

1541

```

1      .title Rz33 Jack of All trades Subroutine
2
3      ;----->
4      ; ROUTINE DESCRIPTION : RZJACK calculates the physical
5      ; address on the disk of the upcoming transfer. In addition, it figures out
6      ; what the DPA address of the upcoming transfer should be based on the virtual
7      ; address that is passed here. If the PPU isn't turned on, the virtual address is
8      ; the physical address as well, and no relocation calculations are needed.
9      ; The Rz33 controller registers are loaded at the same time with the DPA address
10     ; and the physical (sector/track) where we want to go on the disk. These are
11     ; loaded in this routine at the same time because it saves redundant register
12     ; accesses to the Rz33.
13     ; INPUT PARAMETERS : virtual address, LBN number
14     ; OUTPUT PARAMETERS : carry set if seek required, R1 has side 1 value
15     ;----->
16     ; Register Assignments
17
18     ; R0 : DCTRK Image
19     ; R1 : DCEEC Image. R1 has side code on return
20     ; R2 : DDPAR Image. Has Virtual address on entry
21     ; R3 : scratch, restored
22     ; R4 : scratch, restored
23     ; R5 : LBN on entry restored
24     ;----->

```

```

1
2 176304          RELOCATE:
3 176304          PUSH    R0          ; all of these are used
4 176306          PUSH    R2          ; during the routine
5 176308          PUSH    R3          ; all but R1 are restored on return
6 176312          PUSH    R4          ; This is register-intensive but saves
7 176314          PUSH    R5          ; time accessing Floppy registers
8
9 176316 010203          MOV     R2,R3
10 176318 003002          CLR     R2          ; also assures that CARRY IS CLEAR
11 176320 032767 000001 001202 BIT     01,000000 ; is I/O enabled?
12 176322 001425          BEB     BRINIT      ; if not, just go and load registers
13
14 176324 010300          DW16: MOV     R3,R0          ; move it again
15 176326 042700 017777          BIC     017777,R0      ; clear off all but upper 3 bits
16 176328 012701 000005          MOV     05,R1          ; loop constant
17
18 176330 006100          10:  ROL     R0
19 176332 077102          SOB     R1,10          ; zoom ! rotate left 4 times
20
21 176334 042700 177761          BIC     0177761,R0      ; only leave the 4 bit selection of PAR
22 176336 062700 172340          ADD     000par0,R0      ; add 'base' PAR
23 176338 011000          MOV     (R0),R0          ; get content of PAR (the relocate value)
24 176340 012701 000006          MOV     06,R1          ; number of times to shift content (x 100)
25
26 176342 006302          ;-- Begin -->
27 176344 006300          20:  ARL     R2          ; remember, R2 was cleared earlier
28 176346 003302          ARL     R0          ; here we simulate ARLC
29 176348 077104          AOC     R2          ; prep carry if valid
30 176350 077104          30:  SOB     R1,20          ; loop around, and around, and around
31
32 176352          ;-- End -->
33 176354 042703 160000          BIC     0160000,R3      ; remove upper 3 bits of VA
34 176356 000003          ADD     R0,R3          ; new physical address, simulate AOC
35 176358 003302          AOC     R2          ; again propagate carry if it is valid
36
37          ; fall thru to next page of code, where R2= highest 6 bits of 22 bit DMA address
38          ; and R3= low order 16 bits of virtual. Note that when the I/O is OFF we just
39          ; branch to the next page with R2 cleared and R3= 16 bit virtual address.

```

For Internal Use Only

MSC70 ROM Bootstrap X03-01 MACRO V05.03b Wednesday 06-Nov-85 11:12 Page 49

R133 Jack of All trades Subroutine.

```

1 176446 016780 000730      BINIT: MOV    R0,R0
2 176452 000300              SHL    R0
3 176454 105000              CLR    R0
4 176456 150300              B190   R3,R0
5 176460 000300              SHL    R0
6 176462 010067 000714      MOV    R0,R0TRK
7 176466 010067 000714      MOV    R3,R3
8 176470 105001              CLR    R1
9                               ; R0 NEW HAS 'OLD TRACK' AND NEW DPA
10                              ; Write as soon as possible, to let settle
11 176472 000302              SHL    R2
12 176474 052702 100000      B15    001715,R2
13                              ; rotate R2 bits to upper DPA, if any
14                              ; No allowing full 22 bit DPA addresses
15                              ; Now R0 has R0TRK, R1 has R0SEC and R2 has R0DPA
16                              ; Next calculate the new track & sector values for the transfer and GO FOR IT
17 176500 005004      BKPW: CLR    R4
18 176502 071427 000036      DIV    0percyl,R4
19 176506 150402              B190   R4,R2
20 176510 010267 000672      MOV    R2,R0DPA
21 176514 005004              CLR    R4
22 176516 071427 000017      DIV    0persid,R4
23 176522 005205              INC    R5
24 176524 150301              B190   R5,R1
25 176526 010167 000652      MOV    R1,R0SEC
26 176532 005001              CLR    R1
27 176536 001402              TST    R4
28                              BEQ    10
29 176540 012701 000002      MOV    0sidel,R1
30                              ; returned parameter needs 0led into command
31 176544 120002      10:  CPOB    R0,R2
32 176546 001401              BEQ    20
33                              ; compare 'old track' and new
34 176550 000261              SEC
35 176552      20:  POP     R5
36 176554              POP     R4
37 176556              POP     R3
38 176560              POP     R2
39 176562              POP     R0
40 176564 000207      NOJMP: RETURN
41                              ; Now, all registers are set for transfer
42

```


Rx33 Local Interrupt Handler

```

1      .title Rx33 Local Interrupt Handler
2
3      ;-----
4      ; ROUTINE DESCRIPTION :
5      ; We assume that if we get into this routine, that we got there via
6      ; an interrupt from the Rx33. A local location, INTFLB, is defined to always
7      ; be 177777 at entry to this routine. We use a zero for the flag. In case the
8      ; dma engine gets the wrong address and writes over the INTFLB, it is unlikely
9      ; that it would write a zero there. This gives us an extra margin of safety.
10     ;
11     ; INPUT PARAMETERS : INTFLB = 177777
12     ; OUTPUT PARAMETERS : INTFLB = 0;
13     ;-----
14     ; Register Assignments
15     ;
16     ; R0 :
17     ; R1 :
18     ; R2 :
19     ; R3 :
20     ; R4 :
21     ; R5 :
22     ; R6 :
23     ;-----
24
25 176566 005767 000606  RXINH: TST    RXCSR          ; read status register to knock down interrupt
26 176572 005037 000414  RUPFH: CLR    @INTFLB        ; clear to signal interrupts
27 176576 000002          RORTI: RTI          ; and return from interrupt
28
29
30     ;-----
31

```

For Internal Use Only

MSD70 RM Bootstrap X83-01 INCRD V05.03b Wednesday 06-Mar-85 11:12 Page 31

WAIT 33.2us Subroutine

```
1
2
3
4
5
6
7
8 176600 012703 000005
9 176604 077301
10 176606 000207
11
```

.cbrt1 WAIT 33.2us Subroutine

```
←-----→
; This subroutine has the sole purpose of wasting
; 33.2us to make sure that the R33 registers have settled. It also wastes
; R3 which is used for the count and is not restored.
←-----→
```

WAIT: MOV 05,R3

10: SUB R3,10

RETURN

```
; count of 5 in boot rom gives 33.2us delta-T
; audit
←-----→
```

DECDM - Common command interlock & execution routine.

```

1      .cbltl DECDM - Common command interlock & execution routine
2
3
4
5      ROUTINE DESCRIPTION :
6      DECDM is a common routine shared by the hardware tests as well as the read code.
7      The command to be DEcduted is passed in R3. R3 is then used for scratch (timeout).
8      The drive to execute the command on is passed in R4.
9
10     5-Nov-85 Add extra 'TST' instruction in wait loop for correct timeout in C-rw
11     INPUT PARAMETERS : R3 has command to do. R4 has unit (0 or 1)
12     OUTPUT PARAMETERS : R3 clobbered, R4 preserved, carry set if fail
13
14     Register Assignments
15
16     R0 :
17     R1 :
18     R2 :
19     R3 : command word to do. clobbered
20     R4 : unit 0. restored
21     R5 :
22
23
24 176610 000304 DECDM: S4B R4 ; rotate the unit number to the high byte
25 176612 030403 01S R4,R3 ; R3 now is complete with unit ID
26 176614 012737 177777 000414 MOV 0177777,00INTFLB ; set interrupt flag location to expecting interrupt
27 176622 010367 000532 MOV R3,DECDR ; load the command ! here it goes !
28 176626 005003 CLR R3 ; start with 177777-1 for timeout count
29
30 176630 005737 000414 -- Begin --
31 176634 001406 16: TST 00INTFLB ; see if interrupt has occurred yet
32 ; if so you were successful ?
33 176636 005737 176776 TST 00176776 ; This is to pad loop so timeout occurs
34 176642 077306 SOB R3,16 ; with the count, this entire loop times out in 833 ns
35
36 176644 000304 S4B R4 ; Fix R4 back up
37 176646 000261 SEC ; set the error flag... S4B clears carry
38 176650 000207 RETURN ; this is 'bad' exit
39
40 176652 000304 DECDM: S4B R4 ; restore R4 to previous state
41 176654 000207 RETURN ; this is 'good' exit
42
43
44

```

Moving Inversions Test Description

```

1      .tbl Moving Inversions Test Description
2      ;
3      ;*****
4      ; Modified Moving Inversions ROM Test for PDP-11s
5      ;
6      ;TEST STRATEGY: Starting with the memory cleared to zeros, make 'n' passes (n)
7      ; from lowest address to highest address. On each pass, every location is
8      ; read to verify that the last pattern was stored correctly, then the data
9      ; in the location is rewritten to contain an additional 'one'. This is
10     ; continued until each location contains all ones. Now that the memory
11     ; contains all 'ones', make 'n' passes thru the memory from highest address
12     ; to lowest. This time read each location and verify that the previous
13     ; pattern is correct, replace a single '1' with a '0', and reread each
14     ; location to insure that the new pattern was stored. When finished with 16
15     ; passes, the memory contains all zeros again.
16     ;
17     ; (n) 'n' = number of binary bits per location
18     ;
19     ;Test Steps:
20     ; 1) Clear memory to all 'zeros'
21     ; 2) Starting at the LOWEST address, and working UP to the highest, read
22     ; a location and verify that the previous data was stored, replace
23     ; the location's data with a single additional 'one' bit, and reread
24     ; the location to insure that the new pattern was stored correctly.
25     ; This sequence is repeated until, on the final pass, the memory
26     ; contains all ones (16 passes thru the memory).
27     ; 3) Starting now at the HIGHEST memory address and working DOWN, read
28     ; each location and verify that the previous pattern is read cor-
29     ; rectly, rewrite the location's data to contain a single additional
30     ; 'zero' bit, then reread the location to insure that the new pattern
31     ; was stored correctly. Continue until the memory contains all zeros
32     ; (after the 16th pass).
33     ; 4) End of test
34     ;
35     ;FAILURE INDICATIONS: On each of the steps described above as a 'check', failure
36     ; to find the expected value is an error
37     ;
38     ;*****
39     ;-

```

```

1          .sbt11 Moving Inversions Test
2
3          ;+
4          ;*****
5          ;INPUTS:
6          ;   R0,R1,R2,R3 are SCRATCH
7          ;   R4 = Base address of the section of memory under test
8          ;   R5 = Last address of the section of memory under test
9          ;   R6 = Stack Pointer. (WARNING: do not push more than 2 words)
10         ;
11         ;OUTPUTS:
12         ;   C Bit set if test FAILED
13         ;       R0 = failing address
14         ;       R4,R5 unchanged
15         ;   C bit clear if test PASSED
16         ;       R4,R5 unchanged
17         ;       R0 thru R3 undefined
18         ;
19         ;REGISTER USAGE:
20         ;   R0 = Address under test
21         ;   R1 = Data Pattern to check for
22         ;   R2 = Data Pattern to store
23         ;   R4 = Base Address of memory under test
24         ;   R5 = Last Address of memory under test
25         ;   R6 = Stack pointer (WARNING: do not push more than 2 words)
26         ;
27         ;*****
28         ;-
29 176636 MOVINU: ;+
30         ;*****
31         ;   STEP 1: Clear all memory locations to zeros
32         ;*****
33         ;-
34
35 176636 010400 MOV    R4,R0          ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
36 176660 005020 CLR     (R0)+        ;CLEAR A LOCATION(WORD), BUMP ADDRESS BY 2
37 176662 020005 CMP     R0,R5          ;CHECK FOR THE LAST ADDRESS CLEARED
38 176664 101775 BLOS    100          ;BR IF MORE TO CLEAR
39
40         ;+
41         ;*****
42         ;   STEP 2: Starting at the LOWEST address, and working UP to the highest,
43         ;   read each location and verify that the previous pattern is read
44         ;   correctly, rewrite the location's data to contain a single
45         ;   additional one, then reread the location to verify that the new
46         ;   pattern was stored correctly. This sequence is repeated 16
47         ;   times, until the memory contains all 'ones'.
48         ;*****
49         ;-
50 176666 005002 CLR     R2          ;INITIALIZE 'PATTERN TO STORE'
51 176670 010201 200:  MOV    R2,R1          ;REPLACE THE LAST PATTERN WITH THE NEXT TO CHECK FC
52 176672 006302 ASL     R2          ;SET ONE MORE 'ONE' BIT IN R2 (PATTERN TO STORE)
53 176674 005202 INC     R2          ;(USING 'INC' VS. 'SLL' SAVES TIME)
54 176676 010400 MOV    R4,R0          ;SET R0 TO BASE ADDRESS OF MEMORY TO TEST
55 176700 020110 300:  CMP     R1,(R0)        ;CHECK FOR LAST PATTERN STORED SUCCESSFULLY
56 176702 001031 bne     1900          ;br if last pattern is not correct
57 176704 010210 MOVL    R2,(R0)        ;STORE NEXT PATTERN

```


*Swing Inversions Test

```

50 176706 020210      CMP     R2,(R0)      ;CHECK FOR NEW PATTERN
51 176710 001026      bne     1906      ;br if new pattern NOT stored correctly
52 176712 062700 000002 add     R2,R0      ;bump test address by 2
53 176716 020005      CMP     R0,R3      ;CHECK FOR LAST ADDRESS REACHED
54 176720 101767      BLOS    306      ;BRANCH IF MORE ADDRESSES TO TEST
55 176722 005702      tst     R2      ;when R2 = 177777 goto next step
56 176724 100361      bpl     206      ;br if more passes to make
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79 176726 010201      600:  MOV     R2,R1      ;SET R1 TO 'PATTERN TO CHECK FOR'
80 176730 005700      tst     R0      ;(fastest way to clear the 'c' bit)
81 176732 006002      ror     R2      ;add another zero to pattern in R2
82 176734 010500      MOV     R5,R0      ;SET R0 TO HIGHEST MEMORY ADDRESS
83 176736 062700 000002 ADD     R2,R0      ;ADJUST ADDRESS 1st TIME SO WE CAN DECREMENT INITIALLY
84 176742 020140      700:  CMP     R1,-(R0)      ;CHECK FOR LAST PATTERN, DECREMENT ADDRESS BY 2
85 176744 001010      bne     1906      ;br if last pattern not stored correctly
86 176746 010210      MOV     R2,(R0)      ;STORE NEXT PATTERN (ONE ADDITIONAL 'ZERO-BIT')
87 176750 020210      CMP     R2,(R0)      ;CHECK FOR PROPER STORAGE OF NEW PATTERN
88 176752 001005      bne     1906      ;br if new pattern NOT stored correctly
89 176754 020004      CMP     R0,R4      ;CHECK FOR LOWEST MEMORY ADDRESS REACHED
90 176756 101371      BHI     700      ;BR IF STILL NOT AT LOWEST ADDRESS
91 176758 005702      tst     R2      ;when R2 = 0, the test is done
92
93 176762 001361      jclc      ;(side effect of clc instruction)
94
95
96
97
98
99
100
101
102 176764 000207      BNE     600      ;BR IF ANOTHER PASS THRU MEMORY IS REQUIRED
103
104 176766
105 176766 000261
106 176770
107 176770
108
109 176770 000207      rts     PC      ;exit with c bit clear
110
111
112

```

1906: ;(ERROR EXIT)
 sec
 900PK: ; these two labels are dummy vectors preserved for compatibilities' sake
 900PK: ; they both point to 'return instruction' below
 rts PC ;return with c bit set


```
1      ;+
2      ;=====
3      ;
4      ;   END OF P.10C ROM BOOTSTRAP PROGRAM
5      ;
6      ;=====
7      ;-
8
9      ;+
10     ;-----
11     ;
12     ;   'REMAIN' gets defined to be equal to the number of bytes left
13     ;   in the ROM (if positive), or to the number of bytes exceeded
14     ;   (if negative)
15     ;
16     ;   if (ROM size is exceeded by this program)
17     ;   then (issue assembly error)
18     ;
19     ;-----
20     ;-
21
22     000006      remain=loadb+.41      ;compute the 0 of bytes remaining or exceeded
23
24     .if !remain-2
25     .error  ;ROM size exceeded by remain bytes
26     .iff
27     .LAC400-1
28     .word  P10EDT4256.1P10MER      ;low byte=ROM version, High byte=ROM edit
29     .endc
30
31     176776      .LAC400-1
32     000403
33
34     173000      .end  coldst
```

Symbol table

ALLJMP= 000037	DSKPHY 176300	MAXSEC= 000017	RDV00M 175342	SRV = 021024
ALTENT 173036	ENHARD= 000200	MDV00M 174616	RSINT = 000320	SUBM 173574
ANVTAL 174634	ERRAPT 173026	MDV00R 174530	RXBSY = 000001	STEP = 000060
BADMD 174630	ERRSW 175624	MDV00L= 000022	RXCINT 175032	STEPIN= 000120
BADMD= 173000	EXTAB= 000400	MDV00T 174542	RXD0N 175416	STEPON= 000160
BDM = 021000	EXTAB= 000410	MDV00T 174414	RXCSR = 177400	SUPCH= 004000
BITSTK 175362	EXECON 176610	MDV00T 173024	RXDEF 175540	SUPCH= 002000
BIT0 = 000001	EXEDNE 176632	MDV00T 173732	RXDLT = 000004	SWTST 174432
BIT1 = 000002	FLTJMP= 000010	MDV00T 173736	RDFAL 175614	TERMD= 100000
BIT10 = 002000	FLTM = 004000	MDV00M 176636	RDFMD 175422	TESTMD= 172340
BIT11 = 004000	FOO = 001024	METDOK= 100000	RDFIH 176572	TRACK= 000004
BIT12 = 010000	FRICINT= 000330	MTREDA= 001000	RDFILP 176174	TRANSR 175632
BIT13 = 020000	HALJP = 000001	MULTI = 000020	RXINE 176566	T4ER1 175412
BIT14 = 040000	HASH = 000400	NOVBIT= 000004	RXIVEC= 000230	T4ER2 175410
BIT15 = 100000	HBJP = 000002	NOJ0AT 176364	RXJOAT 176344 G	T4ER3 175406
BIT2 = 000004	HBM = 001000	PARSIZ= 000000	RXLP1 176232	T4ER4 175404
BIT3 = 000010	HIFTST= 000040	PARST 174456	RXMR = 177406	T4ER5 175402
BIT4 = 000020	INILJP= 000020	PERCYL= 000036	RXMDY= 000200	T5ER1 176022
BIT5 = 000040	INTFLB= 000414	PERSID= 000017	RX00M = 000000	T5ER2 176020
BIT6 = 000100	INTIDK= 000324	PWEDT= 000001	RXPE = 100000	T5ER3 176016
BIT7 = 000200	INTIDY= 000323	PWNER= 000003	RXPM = 000232	TSKIT 176024
BIT8 = 000400	JMPTAR 173622	PIOFAL= 000021	RXREAD 176132 G	VECSIZ= 004000
BIT9 = 001000	JMPTAR 173636	POSIM 176222	RXREG 174732	VERIFY= 000004
BLKJP= 000004	JMPTAR 173630	PTIME0 174674	RXEST 175722 G	WAIT 176600
BLKSM = 002000	JRTST 174010	PTIME1 174714	RXRTI 176576	MRSEC = 000254
BTYPER 173010	JL1ADC 174020	RDER1 176320	RXSEC = 177404	MTRK = 000360
CHIST 176246	JL1ADD 173644	RDER2 176316	RXTRK = 177402	MPAR0= 172340
CLABIT= 000100	JL100N 174326	RDER3 176314	RX30C 173030	MPAR1= 172342
CHIE = 000002	JL1MTP 173742	RDER4 176312	RX30D 173020	MPAR2= 172344
COLDOT 173000 G	JL1YST 173032	RDER = 000304	RX30M 173022	MPAR3= 172346
CPUFAL 174320	LADADD= 176777	RDER = 000214	RX30Y 173016	MPAR4= 172350
CRCEM= 000010	LCHBIT= 000001	RDER = 000340	SEEK = 000020	MPAR5= 172352
DISQ = 010000	LED = 002000	RDK1 176272	SEEKER= 000020	MPAR6= 172354
DIVTST 174266	LEDBIT= 000010	READBT 176036	SEEK = 000030	MPAR7= 172356
DWATST= 000000	LOADFL= 000023	RECAL = 000010	SELP0 = 000400	WVER0= 177572
DWFIG 176372	LOADIZ= 000010	RECUPH 176770	SELP1 = 001000	WVER1= 177574
DWMIT 176446	LOPTST= 000020	REMAIN= 000006	SEMDPK 176770	WVER2= 177576
DOSEEX 176210	MAXCYL= 000117	RESETS= 000005	SID0 = 000000	WP10CS= 170040
DRV = 000000	MAXLBN= 004502	RUFER= 000020	SID1 = 000002	WVER3= 170042
DRV1 = 000400				

.. ABS. 177000 000 (RM,I,GIL,ABS,DJR)

000000 001 (RM,I,LCL,REL,CDI)

Errors detected: 0

*** Assembly statistics

Mark file reads: 0

Mark file writes: 0

Size of mark file: 8359 Words (33 Pages)

Size of core pool: 14336 Words (56 Pages)

Operating system: RT-11 (Under RTEH-11)

Elapsed time: 00:00:34.00

VSI:RPERC,VSI:RPERC=VSI:RPERC